PSØ8

衡器系统单芯片方案

最终版

临时手册

2008 年 7 月

V0.0.4

acam-messelectronic gmbh
solutions in time

目录

目录 2
1 系统前瞻 4
1.1 介绍 4
1.2 特性 4
1.3 应用 4
1.4 芯片结构图 5
2 特性和规格 6
2.1 管脚分配 6
2.2 管脚描述 6
2.3 绝对最大额定值 9
2.4 标准操作条件 9
2.4.1 电器特性 9
2.5 转换器的测量精度 10
2.5.1 精度 vs. 供电电压 11
2.5.2 应用高质量传感器的转换精度 12
2.6 电流消耗 13
2.7 时序 13
2.7.1 Oscillators 13
2.7.2 SPI-接口 13
2.8 封装信息 14
2.8.1 PAD 分配 14
2.8.2 压焊点位置（仅适用于测试版芯片） 15
2.8.3 QFN56 封装外形 16
2.8.4 QFN64 推荐焊接布线 16
3 中央处理单元 (CPU) 17
3.1 方框原理图 17
3.2 算术逻辑单元 (ALU) 17
3.2.1 累加器 17
3.2.2 标志位 18
3.3 Memory 分配 18
3.3.1 ROM 和 EEPROM 分配 18
3.3.2 RAM 分配 19
3.3.3 RAM 地址指针 19
3.4 设置寄存器 19
3.4.1 配置控制寄存器 19
3.4.2 结果寄存器 27
3.4.3 状态寄存器 28
3.5 指令集 29
4 系统 Reset, 休眠模式和自动配置 47
4.1 上电复位 48
4.2 看门狗复位 48
4.3 在管脚 27 的外部复位 48
4.4 休眠模式 48
5 CPU 时钟产生 48
5.1 看门狗计数器和单一转换计数器 49
6 IO-管脚 49
6.1 配置管脚 49
6.2 输出–写 50
6.3 输入–读 50
7 SPI-接口 50
7.1 I 接口 50
7.2 SPI 时序 50
7.3 SPI-指令 51
7.2.1 RAM 读时序 52
7.2.2 RAM 写时序 52
7.2.3 EEPROM 读进入 / 读保护 52
7.2.4 EEPROM 写进入 53
8 LCD-驱动 53
8.1 基本配置 54
8.2 LCD-电源 54
8.3 LCD 输出驱动配置 55
8.4 LCD 控制 57
8.5 连接原理图 59
8.6 设置各段位置 61
9. 高尖端转换器 63
9.1 测量原理 63
9.2 模式和时序 64
9.2.1 single_conversion（单一转换） 65
9.2.2 stretch（拉伸模式） 65
9.2.3 cytime (Cycle Time) 66
9.2.4 avrate (平均率) 66
9.2.5 模式选择标准 67
9.3 性能设置 67
9.3.1 精度 和 AVRate 67
9.3.2 转换时间/测量刷新率（持续测量模式） 68
9.3.3 转换时间 / 测量频率 (单一转换模式) 69
9.4 连接应变传感器 69
9.4.1 半桥模式（连接如全桥） 69
9.4.2 全桥模式 70
9.4.3 四桥模式 70
9.4.4 惠斯通电桥模式 71
9.4.5 全桥连接成半桥低功耗 71

9.5 充电电容 (Cload) 72
9.6 比较器 72
9.6.1 比较器控制 73
9.7 Rtemp / Rref 74
9.7.1 比较器延迟的纠正 74
9.7.2 温度测量 74
9.7.3 Rtemp 和 Rref 的值 75
9.8 预处理 75
9.8.1 Gain 和 Offset 的温度补偿 76
9.8.2 四驱秤的脚差纠正 76
9.8.3 PS08 本身的 Gain-漂移 76
9.9 PS08 的高精度测量 77
9.10 PS08 与外部单片机连接 78
10 振荡器 78
11 电压测量 78
12 Auto-on 79
13 测量范围 1 79
14 样例电路 80
15 已知的 Bugs 82
最后更改 82
联系方式 83

1 系统前瞻

1.1 介绍

PS08 芯片是尤其为称重衡器所专门设计的超低功耗 SoC（System on Chip）片上系统方案。这个芯片内部集成了 24 位的微处理器并且发扬了 PICOSTRAIN 测量原理的优势。由于此优势，其整个系统的电流消耗，包括传感器本身的电流消耗，可以被减少到一个非常低的水平。PS08 是电池驱动衡器和太阳能驱动衡器的理想选择。这个芯片的精度可以允许通过电池供电系统建立高精度 OIML 校准衡器。

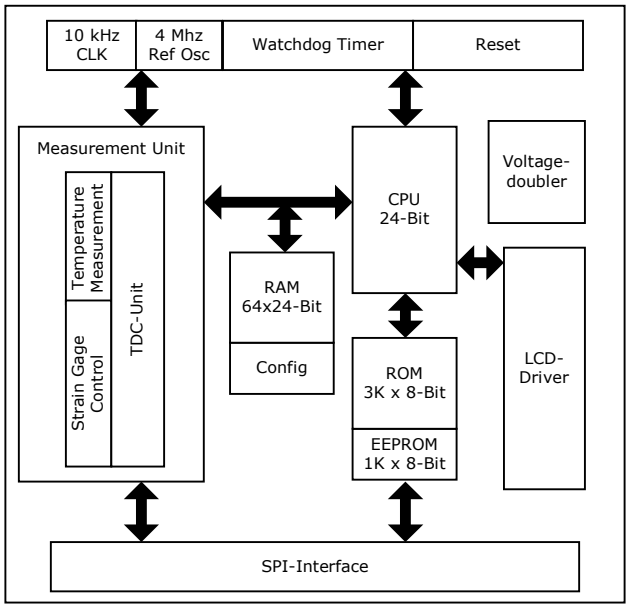


图 1 系统结构

1.2 特性

- PICOSTRAIN 测量尖端最高一百万 (@2mV/V)有效分度，15 万稳定峰峰显示分度
- 带有 24 位数据结构的内部微处理器
- 1 k x 8-位 EEprom 可擦除编程内部存储器
- 3 k ROM 带有强大的软件如 48 位乘法和除法或者 2 进制到 7 段码转换等
- 8-层硬件堆栈
- 超低功耗的嵌入式 10 kHz 振荡器
- 外部 4 MHz 陶瓷晶振驱动
- 静态电流 <1µA
- 5 个可编程 I/O-口
- 4 x 14, 3 x 15, 2 x 16 LCD 驱动
- 驱动 LCD 的嵌入式 charge pump
- 用于电池低压检测的嵌入带隙基准电压
- 带有廉价的碳阻的温度测量断端口
- 看门狗定时器
- 串行 SPI 接口
- 在 130 dB PSRR 电源抑制比时供电电压从 2.2 到 3.6 V
- 系统操作电流最低达到 15 µA
- 可提供裸片 (115 µm 线宽) 或者封装片 (QFN56，7x7 mm2)

1.3 应用

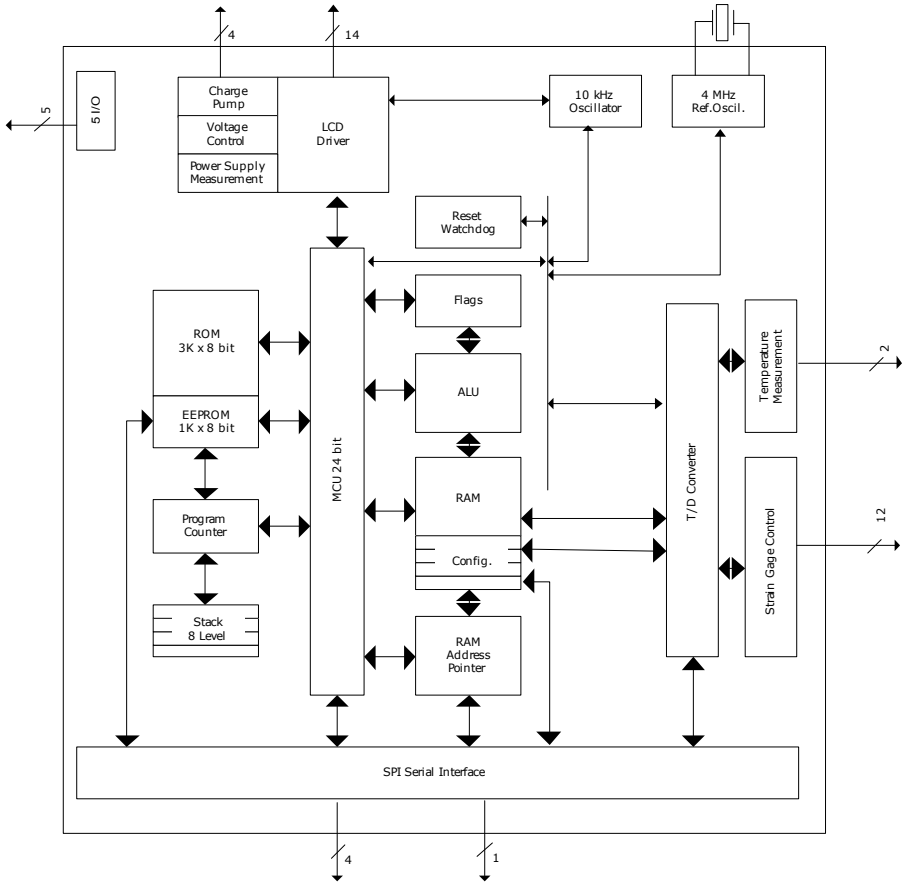
工业及商业方面
- 校准秤
- 计数秤
- 天平

低端消费衡器方面
- 太阳能电源驱动衡器
- 体重秤
- 厨房秤
- 邮局秤
- 包裹秤

重要通知:
本手册为基于工程样片的临时版本。在最终的版本的手册中不排除会有一些主要改变。

1.4 芯片结构图

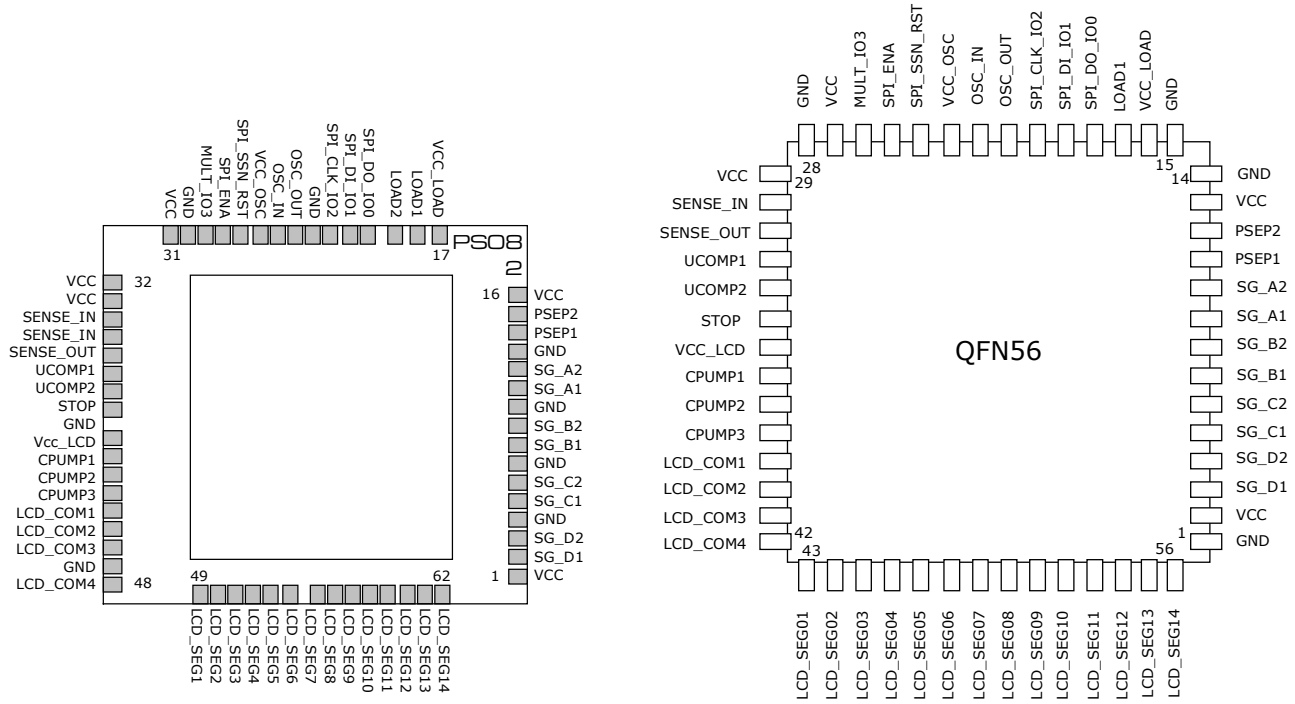
图 1


4
14
5
5 I/O
Charge Pump
Voltage Control
Power Supply Measurement
LCD Driver
10 kHz Oscillator
4 MHz Ref.Oscil.
Reset Watchdog
Flags
ROM 3K x 8 bit
EEPROM 1K x 8 bit
MCU 24 bit
ALU
RAM
Config.
Program Counter
Stack 8 Level
RAM Address Pointer
T/D Converter
Temperature Measurement
Strain Gage Control
2
12
SPI Serial Interface
4
1

2 特性和规格

2.1 管脚分配

图 2



管脚分配

2.2 管脚描述

纯裸片:

#Die	名称	描述	类型	如果不用
1	Vcc	数字部分供电电压, I/O, 4MHz-振荡.		
2	SG_D1	半桥 D 端口 1	N Open Drain	
3	SG_D2	半桥 D 端口 2	N Open Drain	
4	GND	数字部分地端, I/O, Load, LCD, 振荡.		
5	SG_C1	半桥 C 端口 1	N Open Drain	
6	SG_C2	半桥 C 端口 2	N Open Drain	
7	GND	数字部分地端, I/O, Load, LCD, 振荡.		
8	SG_B1	半桥 B 端口 1	N Open Drain	
9	SG_B2	半桥 B 端口 2	N Open Drain	
10	GND	数字部分地端, I/O, Load, LCD, 振荡.		
11	SG_A1	半桥 A 端口 1	N Open Drain	
12	SG_A2	半桥 A 端口 2	N Open Drain	
13	GND	数字部分地端, I/O, Load, LCD, 振荡.		
14	PSEP1	温度测量端口 1	N Open Drain	
15	PSEP2	温度测量端口 2	N Open Drain	
16	Vcc	数字部分供电电压, I/O, 4MHz-振荡		
17	Vcc_load	电源 load 输出管脚 1 和 2		
18	Load1	到测量电容 Load 的 输出	P Open Drain	
19	Load2	到测量电容 Load 的 输出	P Open Drain	
20	SPI_SO_IO0	输出串行 SPI 接口或者 IO-0	Mult-IO	

21	SPI_SI_IO1	输入串行 SPI 接口或者 IO-1	Mult-IO	
22	SPI_SCK_IO2	串行 SPI 接口或者 IO-2 时钟	Mult-IO	
23	GND	数字部分地端, I/O, 充电, LCD,振荡器.		
24	OSC_OUT	4MHz 陶瓷晶振输出		
25	OSC_IN	4MHz 陶瓷晶振输入		
26	Vcc_OSC	4MHz 陶瓷晶振供电电压	Mult-IO	
27	SPI_CSN_RST	从机选择 或者 RST 输入 (高平有效)	Mult-IO 带 pull-down	
28	SPI_ENA	串行 SPI 接口使能	Mult-IO	
29	MULT_IO3	在比较器电路选择惠私通选择器或者 IO3		
30	GND			
31	Vcc			
32	Vcc			
33	Vcc	数字部分地端, I/O, 4MHz-振荡.	Analog In	
34	SENSE_IN	内部 CMOS 比较器输入	Analog Out	
35	SENSE_OUT	内部 CMOS 比较器输出	Analog Out	
36	UCOMP1	外部比较器电路连接	Analog Out	
37	UCOMP2	外部比较器电路连接	Analog In	
38	STOP	测量信号 stop 输入		
39	GND	GND		
40	VCC_LCD	LCD 电源, 10kHz osc.,带隙		
41	CPUMP1	LCD 倍压稳压	Analog Out	
42	CPUMP2	LCD 倍压稳压	Analog Out	
43	CPUMP3	LCD 倍压稳压	Analog Out	
44	LCD_com1	LCD 列驱动 1/2, 1/3, 1/4 duty	LCD Buffer	
45	LCD_com2	LCD 列驱动 1/2, 1/3, 1/4 duty	LCD Buffer	
46	LCD_com3	LCD 行驱动 1/3, 1/4 duty, 列驱动 1/2 duty	LCD Buffer	
47	GND			
48	LCD_com4	LCD 行驱动 1/4 duty, 列驱动 1/2 , 1 /3 duty	LCD Buffer	
49	LCD_seg1	LCD 列驱动	LCD Buffer	
50	LCD_seg2	LCD 列驱动	LCD Buffer	
51	LCD_seg3	LCD 列驱动	LCD Buffer	
52	LCD_seg4	LCD 列驱动	LCD Buffer	
53	LCD_seg5	LCD 列驱动	LCD Buffer	
54	LCD_seg6	LCD 列驱动	LCD Buffer	
55	LCD_seg7	LCD 列驱动	LCD Buffer	
56	LCD_seg8	LCD 列驱动	LCD Buffer	
57	LCD_seg9	LCD 列驱动	LCD Buffer	
58	LCD_seg10	LCD 列驱动	LCD Buffer	
59	LCD_seg11	LCD 列驱动	LCD Buffer	
60	LCD_seg12	LCD 列驱动	LCD Buffer	
61	LCD_seg13	LCD 列驱动	LCD Buffer	
62	LCD_seg14	LCD 列驱动	LCD Buffer	

封装, QFN56:

#QFN	名称	描述	类型	如果不用
1	GND	地		
2	Vcc	数字部分供电电压, I/O, 4MHz-振荡		
3	SG_D1	端口 1 半桥 D	N Open Drain	

4	SG_D2	端口 2 半桥 D	N Open Drain	
5	SG_C1	端口 1 半桥 C	N Open Drain	
6	SG_C2	端口 2 半桥 C	N Open Drain	
7	SG_B1	端口 1 半桥 B	N Open Drain	
8	SG_B2	端口 2 半桥 B	N Open Drain	
9	SG_A1	端口 1 半桥 A	N Open Drain	
10	SG_A2	端口 2 半桥 A	N Open Drain	
11	PSEP1	端口 1 温度测量	N Open Drain	
12	PSEP2	端口 2 温度测量	N Open Drain	
13	Vcc	数字部分供电电压, I/O, 4MHz-振荡		
14	GND	地		
15	GND	地		
16	Vcc_load	电源 load 输出管脚 1 和 2		
17	Load1	Load 输出接测量电容	P Open Drain	
18	SPI_SO_IO0	输出口 串行 SPI 接口 或 IO0	Mult-IO	
19	SPI_SI_IO1	输入口 串行 SPI 接口 或 IO1	Mult-IO	
20	SPI_SCK_IO2	串行 SPI 接口时钟 或 IO2	Mult-IO	
21	OSC_OUT	4MHz 陶瓷晶振输出		
22	OSC_IN	4MHz 陶瓷晶振输入		
23	Vcc_OSC	4MHz 振荡器电源	Mult-IO	
24	SPI_CSN_RST	Spi 片选或 RST 输入(高平有效)	Mult-IO 带 pull-down	
25	SPI_ENA	串行 SPI 接口使能	Mult-IO	
26	MULT_IO3	在比较器电路中选择惠私通选择器或者 IO3 或者中断端口		
27	Vcc			
28	GND	GND		
29	Vcc	数字部分地端, I/O, 4MHz-振荡.	Analog In	
30	SENSE_IN	内部 CMOS 比较器输入	Analog Out	
31	SENSE_OUT	内部 CMOS 比较器输出	Analog Out	
32	UCOMP1	外部比较器电路连接	Analog Out	
33	UCOMP2	外部比较器电路连接	Analog In	
34	STOP	测量信号 stop 输入		
	GND	GND		
35	VCC_LCD	LCD 电源, 10kHz osc.,带隙		
36	CPUMP1	LCD 倍压稳压	Analog Out	
37	CPUMP2	LCD 倍压稳压	Analog Out	
38	CPUMP3	LCD 倍压稳压	Analog Out	
39	LCD_com1	LCD 列驱动 1/2, 1/3, 1/4 duty	LCD Buffer	
40	LCD_com2	LCD 列驱动 1/2, 1/3, 1/4 duty	LCD Buffer	
41	LCD_com3	LCD 行驱动 1/3, 1/4 duty, 列驱动 1/2 duty	LCD Buffer	
42	LCD_com4	LCD 行驱动 1/4 duty, 列驱动 1/2 , 1 /3 duty	LCD Buffer	
43	LCD_seg1	LCD 列驱动	LCD Buffer	
44	LCD_seg2	LCD 列驱动	LCD Buffer	
45	LCD_seg3	LCD 列驱动	LCD Buffer	
46	LCD_seg4	LCD 列驱动	LCD Buffer	
47	LCD_seg5	LCD 列驱动	LCD Buffer	
48	LCD_seg6	LCD 列驱动	LCD Buffer	
49	LCD_seg7	LCD 列驱动	LCD Buffer	

50	LCD_seg8	LCD 列驱动	LCD Buffer	
51	LCD_seg9	LCD 列驱动	LCD Buffer	
52	LCD_seg10	LCD 列驱动	LCD Buffer	
53	LCD_seg11	LCD 列驱动	LCD Buffer	
54	LCD_seg12	LCD 列驱动	LCD Buffer	
55	LCD_seg13	LCD 列驱动	LCD Buffer	
56	LCD_seg14	LCD 列驱动	LCD Buffer	

2.3 绝对最大额定值

符号	参数	条件	最小	最大	单位
Vcc Vcc_load Vcc_osc Vcc_LCD	供电电压	Vcc vs. GND	-0.5	5.0	V
Vin	DC 输入电压		-0.5	Vcc + 0.5	V
Tstg	储藏温度	塑料封装	-55	150	°C

2.4 标准操作条件

符号	参数	条件	最小	最大	单位
Vcc Vcc_load Vcc_osc Vcc_LCD	供电电压	Vcc vs. GND (* 不带 EEPROM 编程以及电压测量的时候)	2.2 (1.5*)	3.6	V
Vin	DC 输入电压		0.0	Vcc	V
Vout	输出电压		0.0	Vdd	V
Top	操作温度		-40	125	°C
Tstg	存储温度	塑料封装	-55	150	°C

2.4.1 电器特性

符号	参数	条件	最小	典型	最大	单位
Vil	输入低电压	CMOS			0.3Vcc	V
Vih	输入高电压	CMOS	0.7Vcc			
Vhyst	输入滞后	Vcc = 3.6 V Vcc = 3.0 V Vcc = 2.7 V Vcc = 2.2 V Vcc = 1.8 V		400 280 225 150 80		mV
Voh	输出高电压		0.8			V
Vol	输出低电压				0.2Vcc	V
Vlbat	低电池电压检测		2.2		2.9	V
LCD_com LCD_seg	LCD 驱动稳压	lcd_vlt = 0 lcd_vlt = 1 lcd_vlt = 2		2.0 2.5 3.0		V

2.5 转换器的测量精度

下面的表格说明了 PSØ8 在不同的电源电压下的测量能力。

表 1 在 Vcc = 3,6 V 时应用外部比较器的测量性能

	ENOB ***dR/R 应变电阻***				精度在 2 mV/V 最大输出, ***快速启动***			
频率 (Hz)	无滤波	SINC3	SINC5		ENOB	分度	噪声 nV	噪声 nV
						有效值	rms	峰-峰值
500	23.8	24.8	25.2		14.8	28,000	231	1,386
250	24.4	25.2	25.7		15.4	44,000	148	891
100	25.2	25.8	26.1		16.2	74,000	89	535
50	25.5	26.2	26.5		16.5	95,000	69	416
20	26.0	26.8	27.0		17.0	133,000	49	297
10	26.6	27.4	27.7		17.6	200,000	33	198
5	27.2	27.9	28.3		18.2	294,000	22	135

	精度在 2 mV/V 最大输出, ***SINC3 滤波***				精度在 2 mV/V 最大输出, ***SINC5 滤波***			
频率 (Hz)	ENOB	分度	噪声 nV	噪声 nV	ENOB	分度	噪声 nV	噪声 nV
		有效位	rms	peak-peak		有效值	rms	峰-峰值
500	15.8	55,000	118	713	16.2	74,000	89	535
250	16.2	74,000	89	535	16.7	105,000	62	376
100	16.8	114,000	57	347	17.1	142,000	46	277
50	17.2	153,000	42	257	17.5	181,000	36	218
20	17.8	222,000	29	178	18.0	266,000	24	149
10	18.4	344,000	19	115	18.7	416,000	15	95
5	18.9	476,000	13	83	19.3	625,000	10	63

表 2 在 Vcc = 3.6 V 应用内部比较器的测量性能

	ENOB ***dR/R 应变电阻***				精度在 2 mV/V 最大输出, ***快速启动***			
频率 (Hz)	无滤波	SINC3	SINC5		ENOB	分度	噪声 nV	噪声 nV
						有效	rms	峰-峰
500	*23.0*	*24.0*	*24.4*		*14.0*			
250	*23.6*	*24.4*	*25.1*		*14.6*			
100	*24.4*	*25.0*	*25.3*		*15.4*			
50	*24.7*	*25.4*	*25.7*		*15.7*			
20	*25.2*	*26.0*	*26.2*		*16.2*			
10	*25.8*	*26.6*	*26.9*		*16.8*			
5	*26.4*	*27.1*	*27.5*		*17.4*			

表 3 整体参数

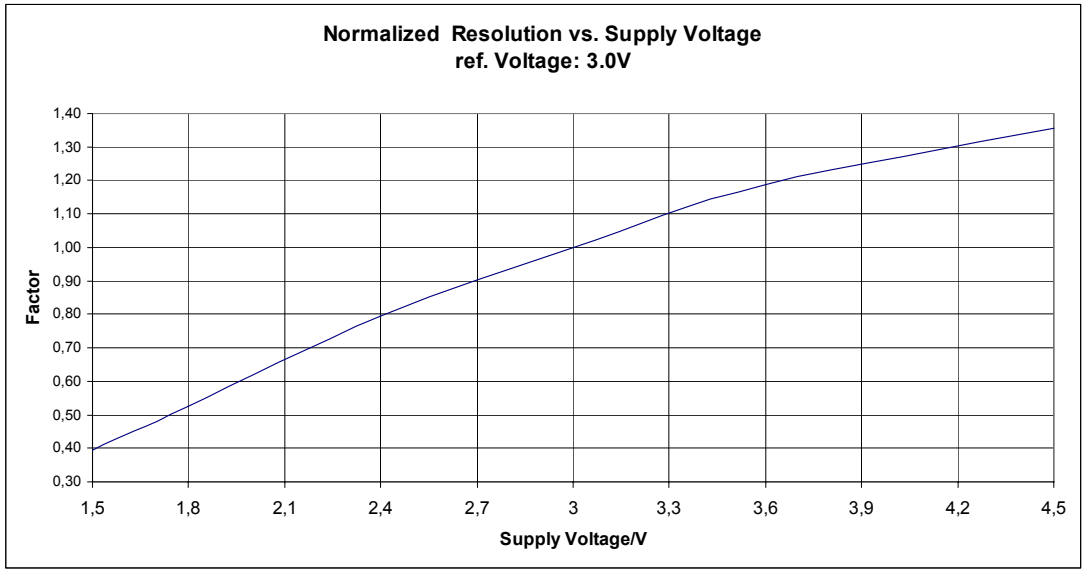
符号	参数	条件	最小	典型	最大	单位
INL	应变积分非线性度					% of FS
	Offset 漂移	整个系统, 1 kΩ DMS, 3V		15		µV
		全桥		15		nV/K
		半桥		< 1		nV/K
	增益漂移	整个系统. 1 KΩ DMS, 5V				ppm/K
	在 0°C ... 70°C 范围					
PSSR1[1]	Vcc 电源抑制比	1.8V 或者 3.3 V +-0.3 V	106 @1.8V	130 @3.3V		dB

*如果应用推荐电路

2.5.1 精度 vs. 供电电压

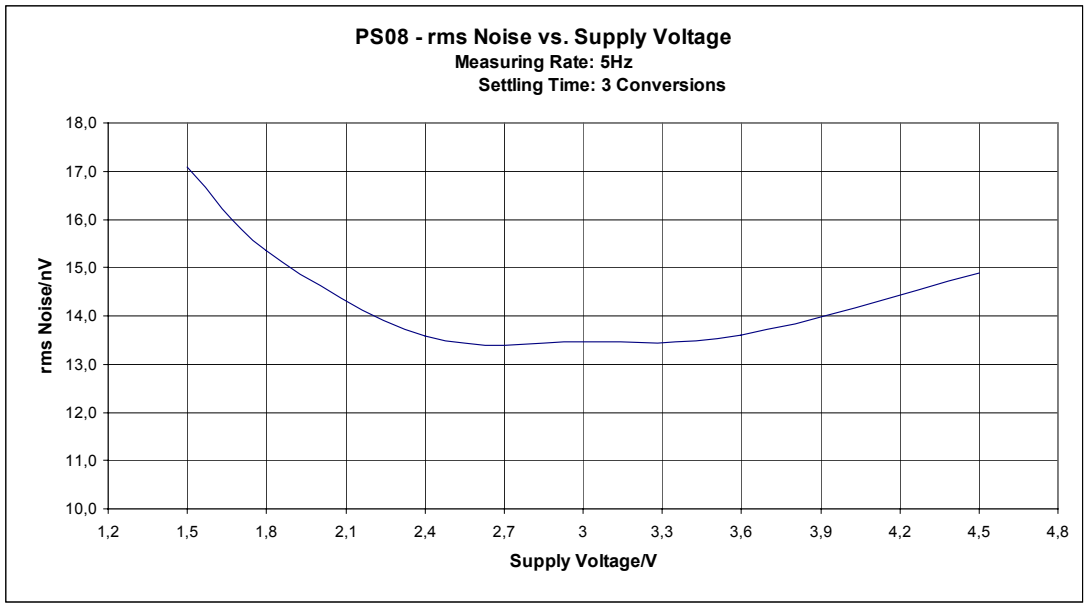
PS08 可以在一个非常大的供电范围内工作。那么可以达到的精度与供电电压有直接关系。电压越高，那么所达到的精度就越高。下面的图说明了供电电压与精度的一个关系，所有数值与 3V 电压为参考：

图 3



下面面的图说明了供电电压与噪声的相关性。最低输入噪声在电压为 2.4 V 和 3.6 V 之间获得。最大搽粉输入电压 [例如 6.6 mV @ 2mV/V 和 3.3 V 供电电压] 除以输入噪声给出了有效精度。

图 4



2.5.2 应用高质量传感器的转换精度

下面的图说明了 PS08 在结合一个 HBM 的高质量传感器情况下的测量值。 这些数据说明了，在应用一个 SINC8 滤波的情况下可以达到 1 百万内部分度，足以实现 1 5 万稳定显示分度。请注意，这些值都是通过一个高质量的传感器所获得。

	快速启动	SINC3	SINC5	SINC8
有效分度	357.000	624.000	811.000	1.012.000
RMS-噪声/nV	20,1	11,5	8,9	7,1
有效位	18,4	19,3	19,6	19,9

图 5

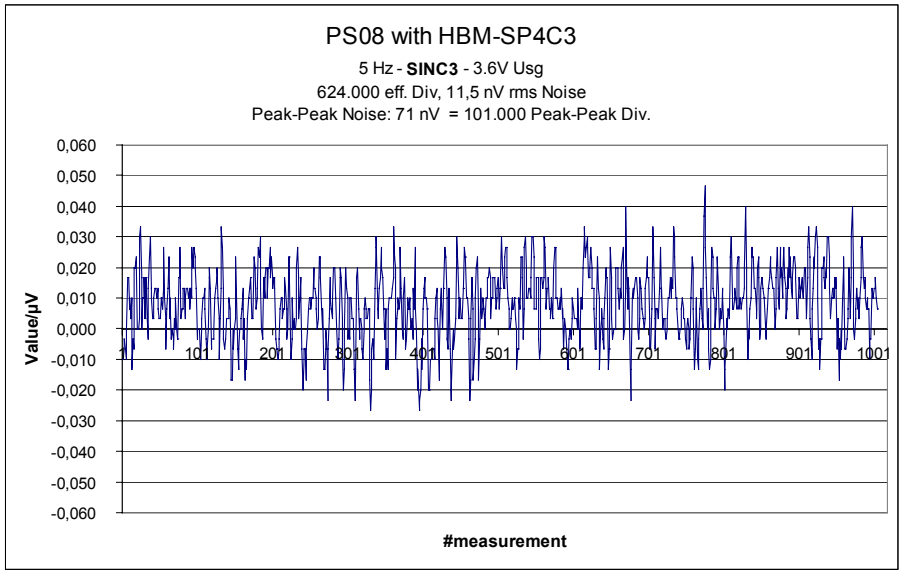
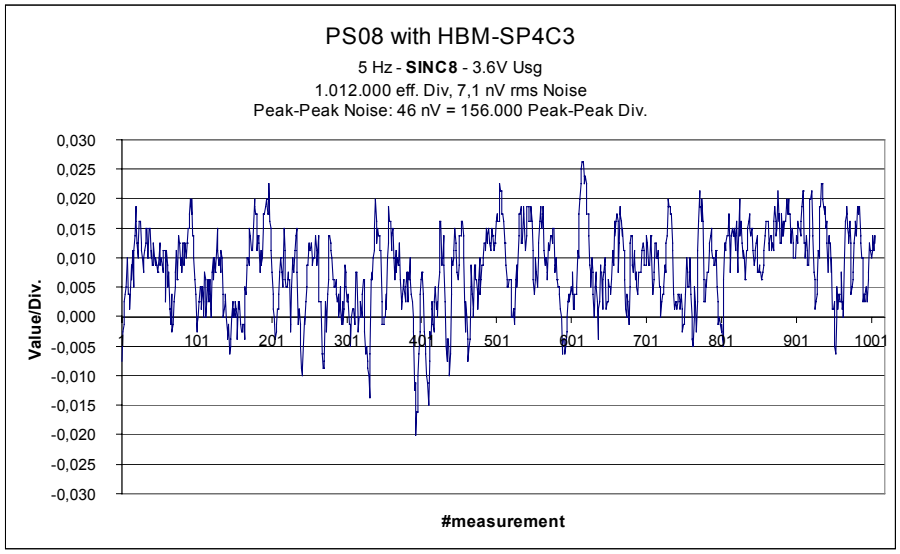


图 6



2.6 电流消耗

下面的表格说明了衡器上包括所有电路器件的电流消耗。

分度值 * *	刷新率	去皮 Tara*	在 3V 时的操作电流		衡器类型	工作时间
2,000	3 Hz	有	1 kOhm	15 µA	太阳能驱动衡器	
2,000	5 Hz	有	1 kOhm 350 Ohm	60 µA 90 µA	邮局称，体重称，厨房称，口袋称。	3,000 小时 (1xCR2032)
5,000	5 Hz	有	1 kOhm 350 Ohm	120 µA 220 µA	中端邮局称, 厨房称，口袋称。	1,500 小时 (1xCR2032)
10,000	5 Hz	有	1 kOhm 350 Ohm	300 µA 700 µA	中端口袋称, 计数称	2,000 小时 (1xCR2430)
80,000	5 Hz	无	1 kOhm 350 Ohm	1.9 mA 4.5 mA	高端口袋称, 计数称	1,500 小时 2 x AA

* 当带有去皮 Tara 功能的时候，如果在满载的时候设置了 Tara,那么将不会有测量量程的溢出。那么可以在衡器上额外的加另一次满量程负载的重量(no = 2 mV/V @ 最大负载)

* *分度值为 5 个标准偏差的峰-峰值 (例如 80.000 分度有效精度实际为 400.000 内码)

2.7 时序

在 Vcc= 3,3 V ± 0,3V, 外界温度为 –40°C ... +85°C 时否则另外说明

2.7.1 Oscillators

表 3 振荡器时序

符号	参数	最小	典型	最大	单位
Clk10kHz	32 kHz 参考振荡时钟		10		kHz
to10st	振荡启振时间				µs
ClkHS	高速参考振荡时钟		4		MHz
toHSst	带陶瓷振荡器的振荡启振时间		50	150	µs

2.7.2 SPI-接口

表 4 串行接口通信时序

符号	参数	最小	典型	最大	单位
fclk	串行时钟频率			1	MHz
tpwh	串行时钟, 脉冲宽度高平	500			ns
tpwl	串行时钟, 脉冲宽度低平	500			ns
tsussn	SSN 开启到 有效时钟沿	500			ns
tpwssn	在写循环之间的 SSN 的脉冲宽度	500			ns
thssn	在 SCLK 下降沿之后的数据保持时间				
tsud	数据准备时间 在 SCLK 下降沿优先	30			ns
thd	在 SCLK 下降沿之前的数据保持时间	30			ns
tvd	在 SCLK 上升沿后数据有效时间				ns

串行接口(SPI 兼容, 时钟相位=1, 时钟极性 =0):

图 7 写时序

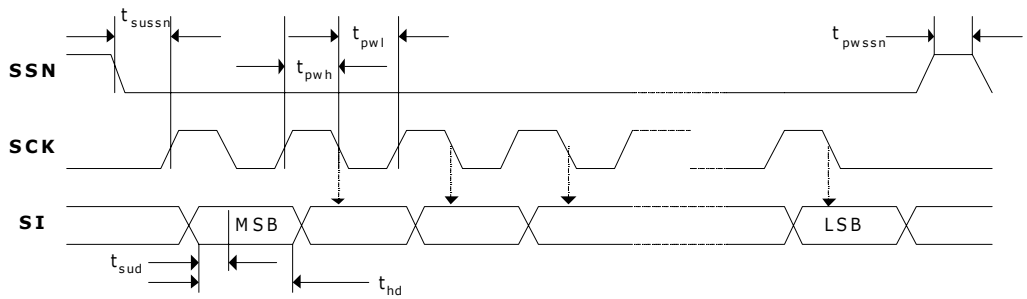
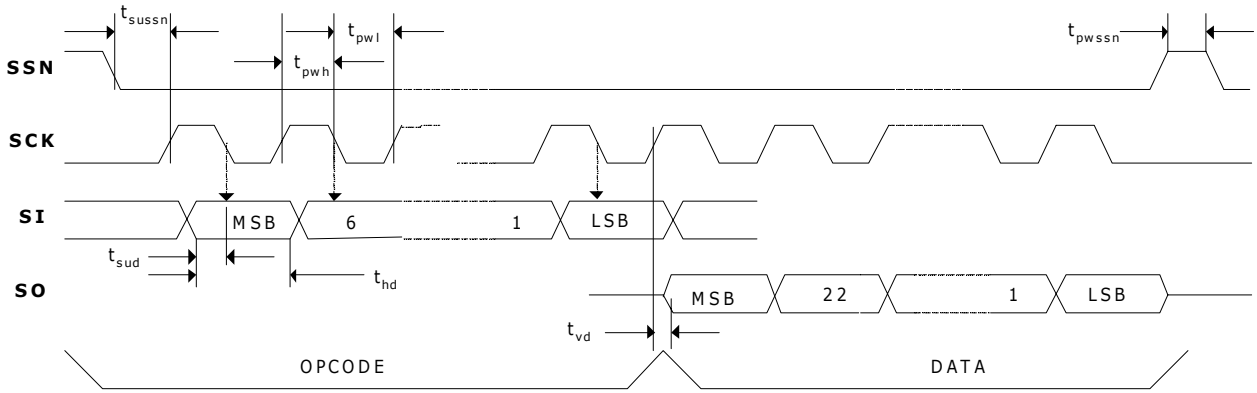


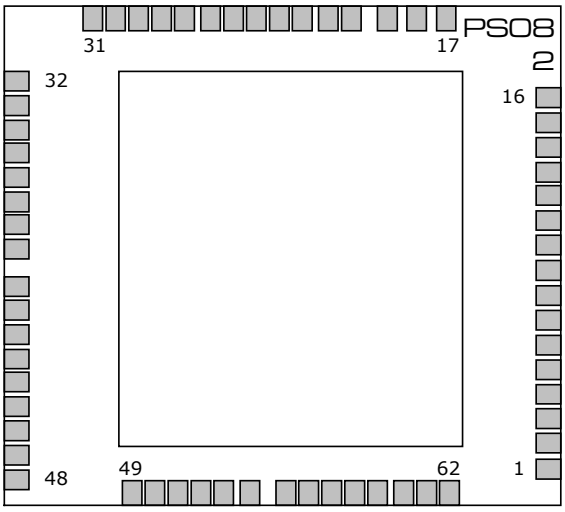
图 8 读时序



2.8 封装信息

2.8.1 PAD 分配

图 9



2.8.2 压焊点位置（仅适用于测试版芯片）

表 6 焊点位置

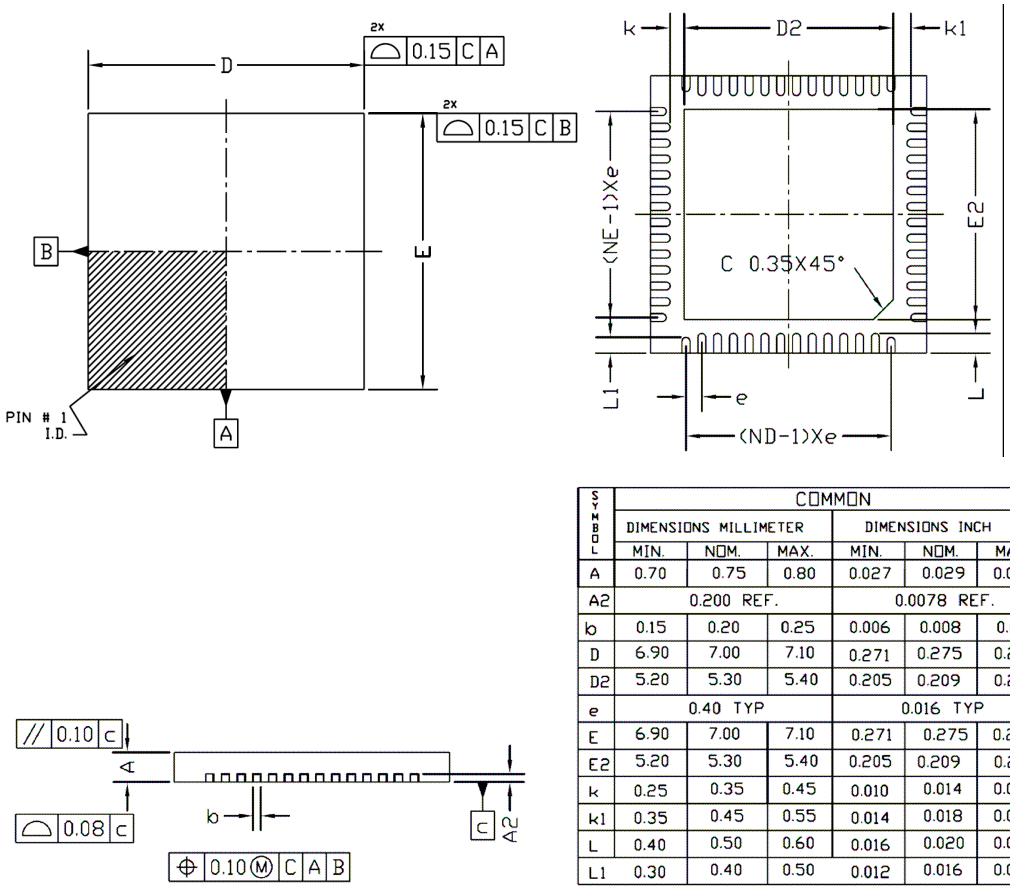
管脚#	名称	X [µm]	Y [µm]	位置	管脚#	名称	X	Y	位置
1	VCC	2583.8	193.8	右	32	VCC	83	2003.3	左
2	SG_D1	2583.8	316	右	33	VCC	83	1891.3	左
3	SG_D2	2583.8	431	右	34	SENSE_IN	83	1779.3	左
4	GND	2583.8	546	右	35	SENSE_OUT	83	1667.3	左
5	SG_C1	2583.8	661	右	36	UCOMP1	83	1555.3	左
6	SG_C2	2583.8	776	右	37	UCOMP2	83	1443.3	左
7	GND	2583.8	891	右	38	STOP	83	1331.3	左
8	SG_B1	2583.8	1006	右	39	GND	83	1219.3	左
9	SG_B2	2583.8	1121	右	40	VCC_LCD	83	1045	左
10	GND	2583.8	1236	右	41	CPUMP1	83	933	左
11	SG_A1	2583.8	1351	右	42	CPUMP2	83	821	左
12	SG_A2	2583.8	1466	右	43	CPUMP3	83	709	左
13	GND	2583.8	1581	右	44	LCD_com1	83	597	左
14	PSEP1	2583.8	1696	右	45	LCD_com2	83	485	左
15	PSEP2	2583.8	1811	右	46	LCD_com3	83	373	左
16	VCC	2583.8	1926	右	47	GND	83	261	左
17	VCC_LOAD	2115	2286	上	48	LCD_com4	83	149	下
18	LOAD1	1976.6	2286	上	49	LCD_SEG1	612.6	83	下
19	LOAD2	1835	2286	上	50	LCD_SEG2	724.6	83	下
20	SPI_SO_IO0	1656.2	2286	上	51	LCD_SEG3	836.6	83	下
21	SPI_SI_IO1	1544.2	2286	上	52	LCD_SEG4	948.6	83	下
22	SPI_SCK_IO2	1432.2	2286	上	53	LCD_SEG5	1060.6	83	下
23	GND	1320.2	2286	上	54	LCD_SEG6	1172.6	83	下
24	OSC_OUT	1208.2	2286	上	55	LCD_SEG7	1347	83	下
25	OSC_IN	1096.2	2286	上	56	LCD_SEG8	1459	83	下
26	VCC_OSC	984.2	2286	上	57	LCD_SEG9	1571	83	下
27	SPI_CSN_RST	872.2	2286	上	58	LCD_SEG10	1683	83	下
28	SPI_ENA	760.2	2286	上	59	LCD_SEG11	1795	83	下
29	MULT_IO3	648.2	2286	上	60	LCD_SEG12	1907	83	下
30	GND	536.2	2286	上	61	LCD_SEG13	2019	83	下
31	VCC	424.2	2286	上	62	LCD_SEG14	2131	83	下

PAD 号：56 脚，90 µm 宽，116 µm 高; 裸片尺寸：2770x2520 μm^2

2.8.3 QFN56 封装外形

QFN56, 7x7 mm², 0.4mm 线宽

图 10



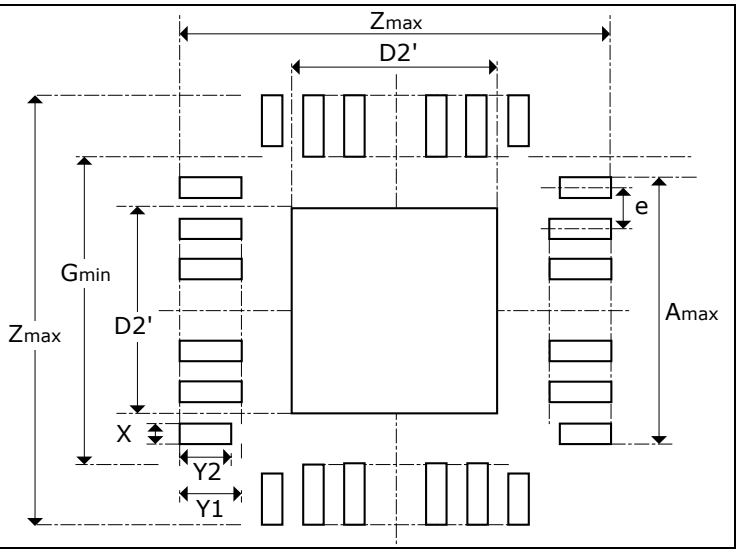
SYMBOL	COMMON					
	DIMENSIONS MILLIMETER			DIMENSIONS INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	0.70	0.75	0.80	0.027	0.029	0.031
A2		0.200 REF.			0.0078 REF.	
b	0.15	0.20	0.25	0.006	0.008	0.010
D	6.90	7.00	7.10	0.271	0.275	0.279
D2	5.20	5.30	5.40	0.205	0.209	0.213
e		0.40 TYP			0.016 TYP	
E	6.90	7.00	7.10	0.271	0.275	0.279
E2	5.20	5.30	5.40	0.205	0.209	0.213
k	0.25	0.35	0.45	0.010	0.014	0.018
k1	0.35	0.45	0.55	0.014	0.018	0.022
L	0.40	0.50	0.60	0.016	0.020	0.024
L1	0.30	0.40	0.50	0.012	0.016	0.020

2.8.4 QFN64 推荐焊接布线

QFN56

		mm	inch
e	=	0.4	0.016
Gmin	=	6.3	
Zmax	=	8.0	
D2'	=	5.4	
Amax	=	5.45	
X	=	0.25	
Y1	=	0.85	
Y2	=	0.75	

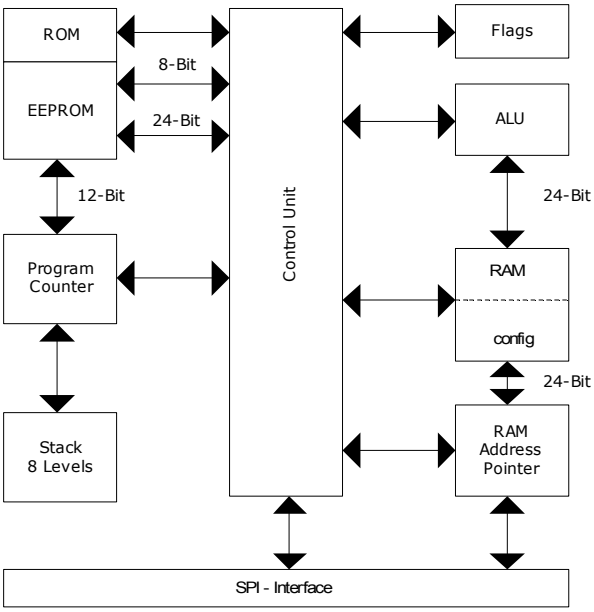
注: 地的面积不能够减少。 不能带有硅通道



3 中央处理单元 (CPU)

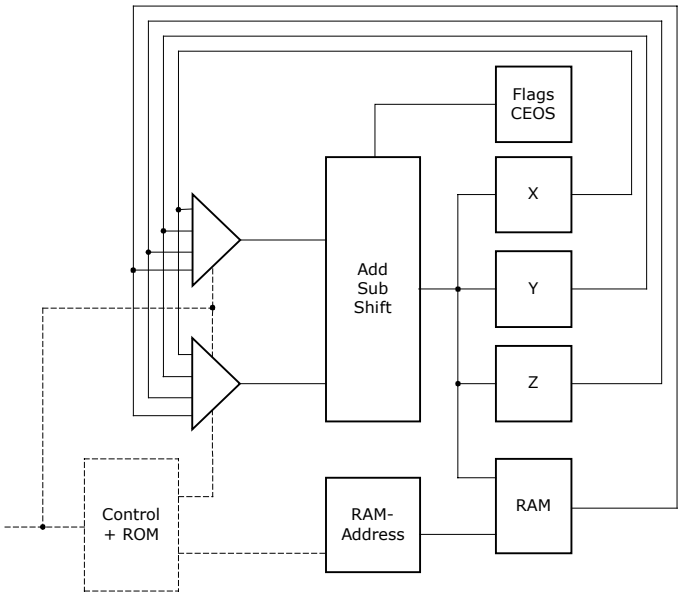
3.1 方框原理图

图 11



3.2 算术逻辑单元 (ALU)

图 12



3.2.1 累加器

ALU 有 3 个 24 位的累加器， A，B 和 C。RAM 是通过 RAM 地址指针寻址的，这个寻址 RAM 单元被用作第四个累加器。通过 RAM 地址指针一个单一的 RAM 地址被配给 ALU。因此总共有 4 个累加器。所有的传输操作（move, swap）和算术运算操作 [shift, add, mult...] 可以应用在所有累加器上。

3.2.2 标志位

处理器对于每个操作都控制 4 个标志位。在对每一个累加器（包括 RAM）读操作时 不-等于（not-equal）和符号标志会被置位。另外, 进位和溢出标志位在计算的情况下被置位(Add/Sub/shiftR)。可以通过 jump 命令来跨越每个标志。

-进位

显示了在一个加法或减法中的进位。与移位操作一起时 (shiftL, rotR 等) 它显示的为延迟位。

-非零

这个标志当有新的结果为 0 写到累加器(add,sub,move,swap 等)的时候被置位。

-符号

这个符号标志当有新的结果被写到累加器(add,sub,move,swap 等)以及最高位(MSB)为 1 时被置位。

-溢出

说明了 2 进制的两个数在加法或减法时出现的溢出。

3.3 Memory 分配

3.3.1 ROM 和 EEPROM 分配

表 1

4095 ... 1024	Program Memory ROM
1023 ... 48	Program Memory EEPROM bank 1 EEPROM bank 2 Program entry
47...45 44...42 41...39	Config Reg 15 (mirrored) Config Reg 14 (mirrored) Config Reg 16 (mirrored)
38...35 5...3 2...0	Config Reg 12 (mirrored) Config Reg 1 (mirrored) Config Reg 0 (mirrored)

ROM 的区域是从 1024 地址开始的。其中包括所有 PICOSTRAIN 测量方法所需要的操作精简指令。还有另一些函数是在衡器应用中经常用到的，例如： 十进制到 7 段码转换等。这些指令可以通过在 EEProm 中的程序来调用。当进行配置时程序也可以从 ROM 跳回到 EEProm。
EEProm 为 1024 Bytes 大小. 用户的程序内存从地址 48 开始占用了 976 bytes. 每一次从 ROM 到 EEProm 的跳转都从地址 48 开始。在 EEProm 中的程序可以通过寄存器 22 中的状态信息来找出跳转的原因。
在 EEProm 中的低 48 字节将会被保留作为 PS08 一次上电复位后自动的寄存器配置。每 3 个连续的字节构成一个寄存器。因此有 16 个 24 位的寄存器可以通过程序代码读出。

低 33 bytes 从 0 到 38 构成 24 bit 中的 13 个语句，用于配置寄存器。在一次上电复位后他们将被拷贝到 RAM 地址 48-60 当中。

EEprom 单元 39 到 47 在标准配置中并不是必须的。处理器可以在操作(putEpr 和 gerEpr)的时候从这些单元中写入或者读出。它们可以被用于存储校准后的数据。

3.3.2 RAM 分配

表 2

64 48	Config Reg 16 ... Config Reg 0
47 ... 32	用户 RAM 47 ... 用户 RAM 32
31 ... 26 25 24 23 22 21 20 19 18 17 16	系统 RAM ... 系统 RAM UBATT CAL HB1+ Flags (p1-p2)/p2 HB0 = (A-B)+(...) / (A+B)+(...) HB4 = (G-H)/(G+H) HB3 = (E-F)/(E+F) HB2 = (C-D)/(C+D) HB1 = (A-B)/(A+B)
15 ... 0	User RAM 15 ... User RAM 0

A..F = 在不同端口的放电时间, 详见 *3.4.2 结果寄存器*

3.3.3 RAM 地址指针

RAM 带有自己的 64 地址的地址总线。24 位宽度是与 ALU 的寄存器宽度相关的。通过 RAM 地址指针可以将一个单一的 RAM 地址指定到 ALU。 然后这个单一的地址将作为第 4 个累加器。改变 RAM 地址指针不会影响到地址 RAM 中的内容。RAM 地址指针通过不同的操作码来改变 (ramadr, incramadr,...)

3.4 设置寄存器

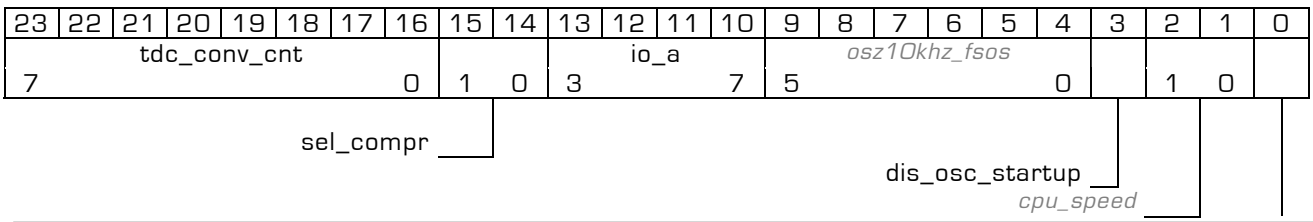
3.4.1 配置控制寄存器

PS08 有 16 个 24 位宽的配置寄存器，在 RAM 中被分配在地址 48 到 63。配置寄存器控制了整个芯片包括了应变的测量和 LCD 控制。

可以向配置寄存器写入数值

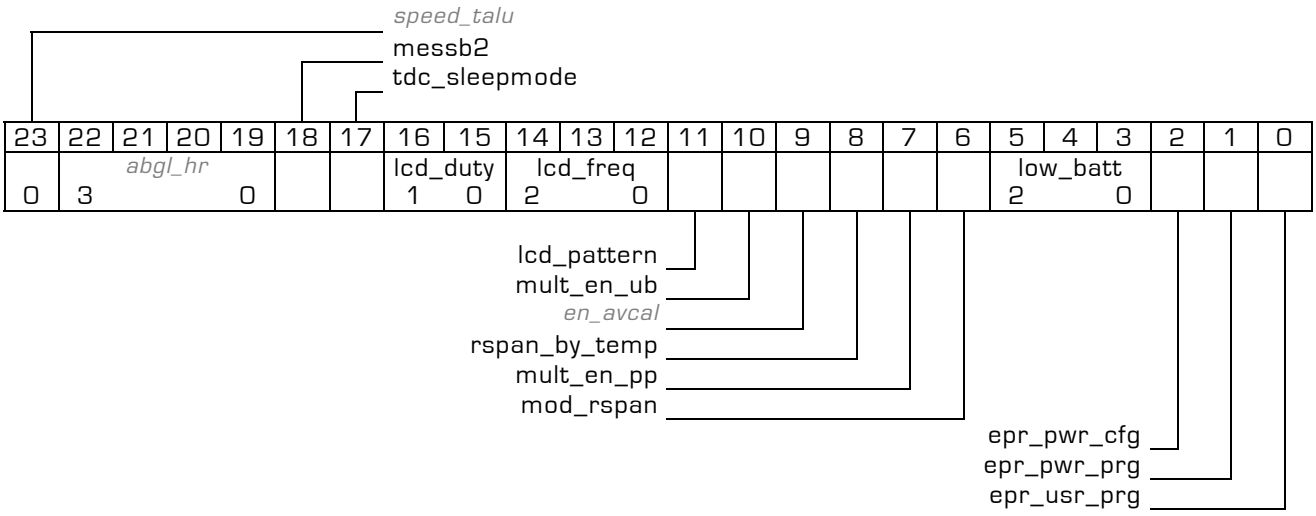
- 通过内部单片机的操作
- 通过外部处理器的 spi 串行通信
- 在上电复位时通过从 EEPROM 中传输默认的寄存器配置值

Configreg_00: RAM 地址 48 EEPROM 字节 0 - 2

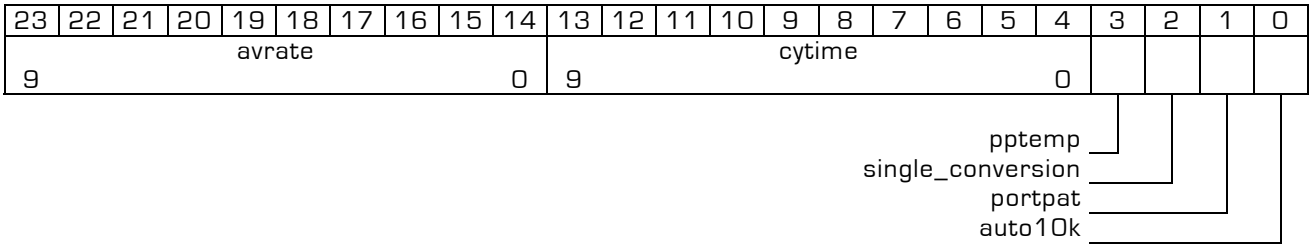


dis_haltpp_ps

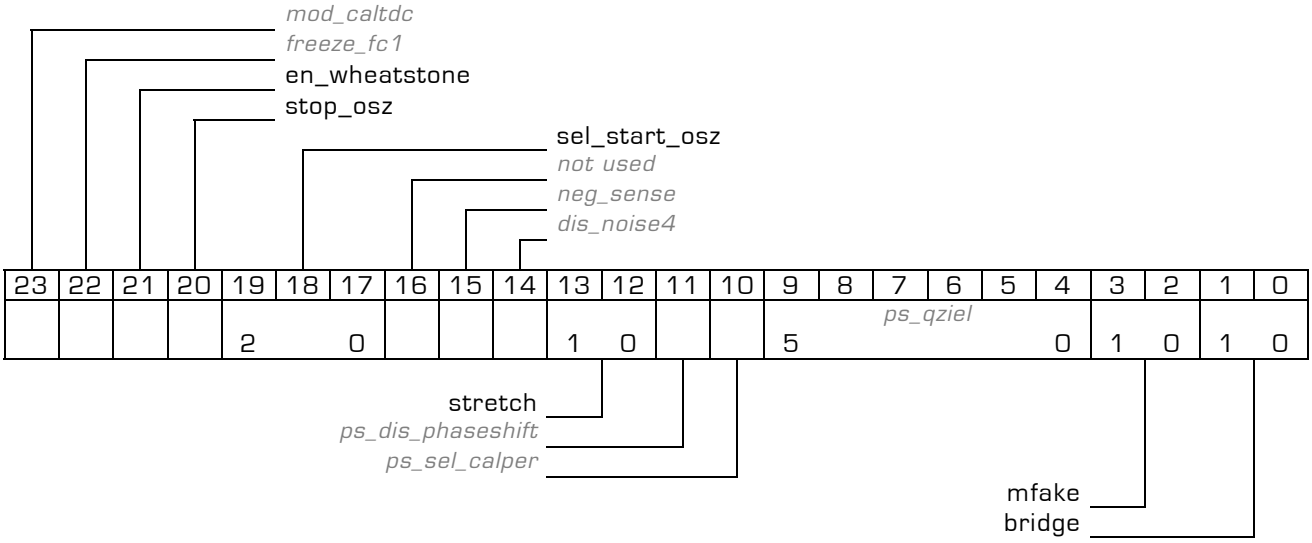
Configreg_01: RAM 地址 49 EEPROM 字节 3 - 5



Configreg_02: RAM 地址 50 EEPROM 字节 6 - 8



Configreg_03: RAM 地址 51 EEPROM 字节 9 - 11



Configreg_04: RAM 地址 52 EEPROM 字节 12 - 14

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mult_Hb1																							
23																							0

Configreg_05: RAM 地址 53 EEPROM 字节 15 - 17

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mult_Hb2																							
23																							0

Configreg_06: RAM 地址 54 EEPROM 字节 18 - 20

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mult_Hb3																							
23																							0

Configreg_07: RAM 地址 55 EEPROM 字节 21 - 23

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mult_Hb4																							
23																							0

Configreg_08: RAM 地址 56 EEPROM 字节 24 - 26

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mult_TkG																							
23																							0

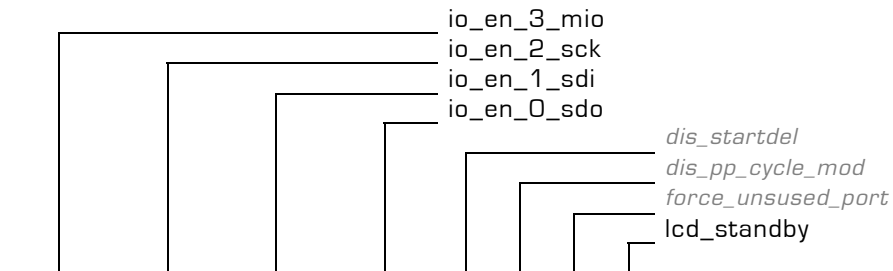
Configreg_09: RAM 地址 57 EEPROM 字节 27 - 29

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mult_TkO																							
23																							0

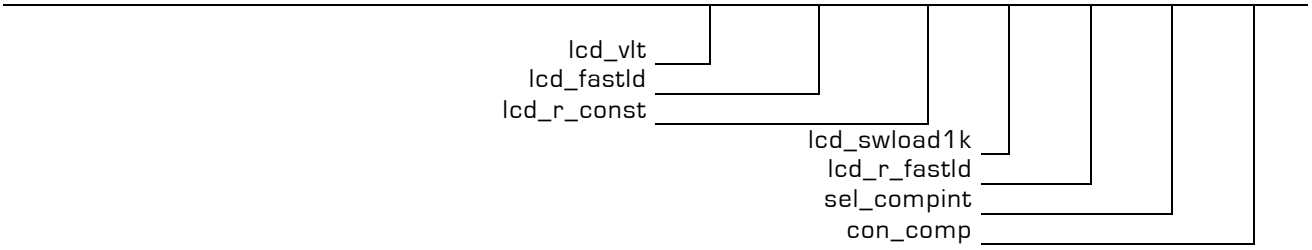
Configreg_10: RAM 地址 58 EEPROM 字节 30 - 32

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
calcor								Mult_Ub								Mult_PP							
7							0	7							0	7							0

Configreg_11: RAM 地址 59 EEPROM 字节 33 - 35



23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0



Configreg_12: RAM 地址 60 EEPROM 字节 36 - 38

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
lcd_pos																							
23																							0

Configreg_13: RAM 地址 61 不在 EEPROM 中被显示！

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
lcd_segment																							
23																							0

Configreg_14: RAM 地址 62 EEPROM 字节 42 - 44

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
lcd_segment																							
47																							24

Configreg_15: RAM 地址 63 EEPROM 字节 45 - 47

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1'bx																lcd_segment							
15															0	55							48

Configreg_16: RAM 地址 64 EEPROM 字节 39 - 41

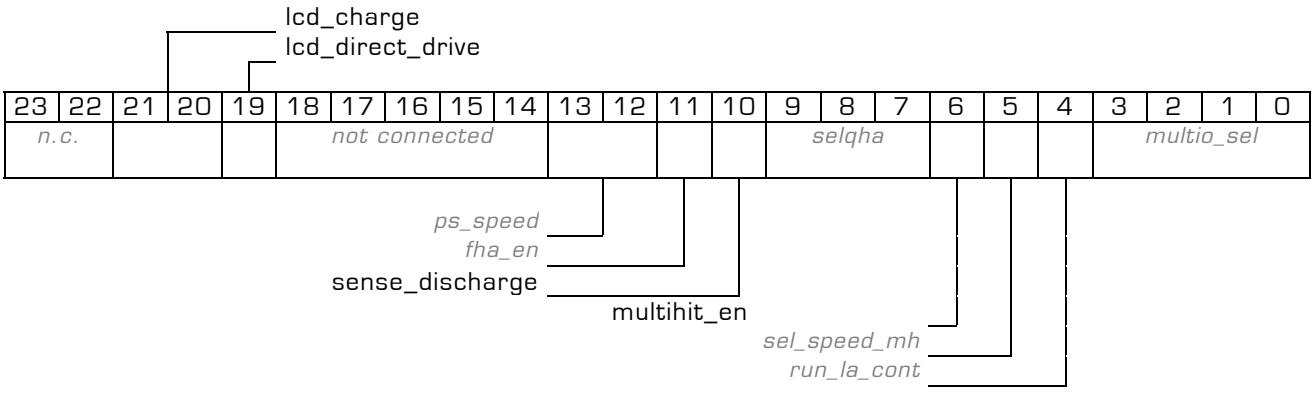


表 3

配置 数值	寄存器	推荐 默认值	硬件 默认值	描述
abgl_hr[3:0]	*1*	*5*	*5*	*高精度调整*
auto10k	2	1	1	通过 4 M时钟每0.6秒自动校准一次 10kHz 时钟. 不能够 stretched-单一转换模式中应用
avrate[9:0]	2	25	1	内部平均率
bridge[1:0]	3	0	0	0 = 不合理 (一个半桥) 1 = 2个半桥 2 = 不支持 3 = 4个半桥
calcor[7:0]	*10*	*0*	*0*	*用于TDC校准的校准系数* *cal := cal + calcor/8 = cal + [-127 to +128]/8* *= cal + [-15.875 to +16.000]*
con_comp[1:0]	11	3	3	比较器关闭模式 00 =关闭 01 = 在两次测量之间的时候关闭 10 = 在两次测量之间以及充电情况下关闭 11 = 一直开启
cpu_speed[1:0]	*0*	*3*	*3*	*CPU环振速度设置* *00 = 快* *01 = 默认* *10 = 慢* *11 = 非常慢*
cytime[7:0]	2	100	100	以 2 µs为倍数的Cycle time (8 * 4 MHz 周期, stretch = 0) 或者 100 µs (10 kHz 周期, stretch = 1)
特殊内部位	*0*	*0*	*0*	*应用默认值*
dis_noise4	*3*	*0*	*0*	*关闭主要噪声单元*
dis_osc_startup	0	1	0	当开启振荡器时减小功耗
dis_pp_cycle_mod	*11*	*0*	*0*	*关闭带有两个cycle的增益测量*
dis_startdel	*11*	*1*	*1*	*关闭start延迟*
en_avcal	*1*	*0*	*0*	*开启 16次平均后的TDC校准值*
en_wheatstone	3	0	0	开启惠斯通电桥模式
epr_pwr_cfg	1	0	0	作为 frontend := 0 stand-alone := 1 在EEprom中的配置将在上电后应用
epr_pwr_prg	1	0	0	作为frontend := 0 stand-alone := 1 在上电复位后在EEprom 中的 PowrOnResetcode 开始运行, 从地址48开始

配置 数值	寄存器	推荐 默认值	硬件 默认值	描述
epr_usr_prg	1	0	0	作为frontend := 0 stand-alone := 1 在上电复位后 在 EEprom中的用户代码开始有效
Special internal Bits	*16*	*0*	*0*	*应用默认值*
force_unused_port	*11*	*0*	*0*	*将不用的测量端口接地*
Special internal Bits	*3*	*0*	*0*	*应用默认值*
io_a[3:0]	0	0	0	I/O's 输出: 输出值, 可以被读回 输入:读输入值
io_en_0_sdo[1:0]	11	3	2	端口定义 00 =输出 01 =带 pull-up的输入 10 =带pull-down的输入 11 =输入
io_en_1_sdi[1:0]	11	3	2	端口定义 00 =输出 01 =带 pull-up的输入 10 =带pull-down的输入 11 =输入
io_en_2_sck[1:0]	11	3	2	端口定义 00 =输出 01 =带 pull-up的输入 10 =带pull-down的输入 11 =输入
io_en_3_mio[1:0]	11	3	2	端口定义 00 =输出 01 =带 pull-up的输入 10 =带pull-down的输入 11 =输入
lcd_duty[1:0]	1	0	0	LCD duty cycle选择 0 = 关闭 1 = 1/2duty 2 = 1/3duty 3 = 1/4duty
lcd_charge[1:0]	16	0	0	再充电前选择 LCD cycle数 0 = 每个cycle充电 1 =每 2 个cycle充电 2 =每 3 个cycle充电 3 =每 4 个cycle充电
lcd_directdrive	16	0	0	直接用电源电压驱动 LCD
lcd_fastld[1:0]	11	2	2	配置快速充电周期数 (10ms)带有低阻分压器
lcd_freq[2:0]	1	4	0	选择 LCD 频率 (LCD像素的开启时间) 像素 | 时间 | Multiplex 模式 1/4 | 1/3 | 1/2 | Hz

配置 数值	寄存器	推荐 默认值	硬件 默认值	描述
				0 8.0ms 15 20 31 Hz 1 4.8ms 26 34 52 Hz 2 4.0ms 31 42 62 Hz 3 3.2ms 30 52 78 Hz 4 2.4ms 52 69 104 Hz 5 2.0ms 62 83 125 Hz 6 1.6ms 78 104 176 Hz 7 1.2ms 104 138 208 Hz
lcd_pos[23:00]	12	o76543210	o76543210	LCD 各段位置
lcd_r_const[1:0]	11	1	0	定义了LCD 分压的交互电阻阻值 0 = 15 k 1 = 200 k 2 = 800 k 3 = 1600 k
lcd_r_fastld[1:0]	11	0	0	选择快速加电LCD位素的电阻值 0 = 15 k 1 = 200 k 2 = 800 k 3 = 1600 k
lcd_segment[23:0]	13	h000000	h000000	显示段码的数字 2 到 0
lcd_segment[47:24]	14	h000000	h000000	显示段码的数字5 到 3
lcd_segment[55:48]	15	h000000	h000000	显示段码上的特殊字符 (1/3 和 1/4 duty)
lcd_standby	11	0	0	0 = LCD 启动 1 = LCD 电源休眠
lcd_swload1k	11	1	1	LCD 驱动电压加倍器用1K 欧姆--代替200 欧姆
lcd_vlt[1:0]	11	1	1	LCD 电压 0 = 2 V 1 = 2.5 V 2 = 3 V 3 = 2 V 不带充电泵
low_batt[2:0]	1	0	0	为低电池电源检测和EEpromwrite 设置电压水平2.2 V, 2.3 V, 2.4 V 到 2.9 V (2.2 V + 0.1 V * low_batt)
messb2	1	0	0	1 =设置 TDC 测量范围2
mfake[1:0]	3	0	0	设置伪测量数目
Special internal bits	*3*	*0*	*1*	*应用推荐值*
mod_rspan	1	0	0	1 =使能开启内部gain补偿电阻Rspan的乘法系数
mult_en_pp	1	0	0	1 =使能gain校准的乘法因数
mult_en_ub	1	0	0	1 =使能电源电压校准的乘法因数
Mult_Hb1[23:0]	4	h100000	h100000	HB1结果的乘法因数 h1 := h1 * [-2^23 to 2^23-1]/ 2^20
Mult_Hb2[23:0]	5	h100000	h100000	HB 2 结果的乘法因数 h1 := h1 * [-2^23 to 2^23-1]/ 2^20
Mult_Hb3[23:0]	6	h100000	h100000	HB 3 结果的乘法因数

配置 数值	寄存器	推荐 默认值	硬件 默认值	描述
				h1 := h1 * [-2^23 to 2^23-1]/ 2^20
Mult_Hb4[23:0]	7	h100000	h100000	HB 4 结果的乘法因数 h1 := h1 * [-2^23 to 2^23-1]/ 2^20
multio_sel[3:0]	*16*	*12*	*0*	*应用 multio03 管脚做诊断* *0 = multio* *1 = clk10khz* *2 = clkalu* *3 = load* *4 = epr_acc* *5 = portin_or* *6 = eprom read access* *7 = testo_tdc* *8 = phaseschifter out* *9 = start_stop* *10 = sense_ac1_comp2* *11 = sense_schmitt_trigger* *12 = interrupt*
High Resolution	*16*	*1*	*1*	*开启高精度应用，不能够用1.5V电源供电*
Mult_PP[7:0]	10	h80	h80	Gain补偿的乘法因数 g := g * [0 to 255]/2^7
Mult_TkG[23:0]	8	h100000	h100000	Rspan 校准的放大系数 Rs := Rs * [-2^23 to 2^23-1]/ 2^20
Mult_TkO[23:0]	9	h000000	h000000	Rspan 的Offset 值, 直接去除
Mult_Ub[7:0]	10	h00	h80	通过电压测量获得gain校准的乘法因数
Special internal bit	*3*	*0*	*0*	*使用默认设置*
osz10khz_fsos[5:0]	*0*	*h20*	*h20*	*10 kHz振荡的频率调整*
portpat	*2*	*1*	*0*	*端口开关模式.强烈土建使用 'On'*
pptemp	*2*	*0*	*0*	*使能gain误差和温度测量*
Special internal bit	*3*	*0*	*0*	*使用默认设置*
ps_qziel[5:0]	*3*	*33*	*17*	*使用推荐值*
Special internal bit	*3*	*0*	*0*	*使用默认设置*
Special internal bit	*16*	*0*	*0*	*使用默认设置*
rspan_by_temp	1	0	0	应用温度测量方法来代替温度补偿中的Rspan补偿
Special internal bit	*16*	*0*	*0*	*使用默认设置*
sel_compint	11	0	0	1 =选择内部比较器
sel_compr	0	0	0	选择比较器工作电阻 00 = 10k 01 = 10k 10 = 7k 11 = 4.1k
Special internal bit	*16*	*2*	*2*	*使用默认设置*
Special internal bit	*16*	*0*	*0*	*使用默认设置*
sel_start_osz[2:0]	3	0	0	设置在开始测量时4 MHz晶振的延迟时间

配置 数值	寄存器	推荐 默认值	硬件 默认值	描述
				0 = off 1 = 持续开启 2 = 100 µs 3 = 200 µs 4 = 300 µs 5 = 400 µs 6 & 7 不连接
sense_discharge	16	1	0	设置比较器低通电容的快速放电
single_conversion	2	0	0	0 = 持续测量模式 1 = 单一转换模式
speed_talu	*1*	*0*	*0*	*使用默认设置*
stop_osz	3	0	0	通过命令停止振荡 (例如在 AutoOn 后没有中断)
stretch	3	0	0	选择 stretched 模式 0 = 关闭 1 = 单一 R 的测量 2 = 2xR (半桥), 200 µs 延迟 3 = 2xR (半桥), 300 µs 延迟
tdc_conv_cnt[7:0]	0	0	0	基于 10 kHz/64 = 156.25 Hz 的单一转换定时器
tdc_sleepmode	1	25	25	不带 T D C 和应变测量的扫描模式，用于当衡器关闭时扫描是否有重量，同于 avrate=0

3.4.2 结果寄存器

在一次测量结束后 RAM 结果寄存器的内容:

ram=16 : HB1=(A-B) / (A+B) HB1 非补偿情况
ram=17 : HB2=(C-D) / (C+D) HB2 非补偿情况
ram=18 : HB3=(E-F) / (E+F) HB3 非补偿情况
ram=19 : HB4=(G-H) / (G+H) HB4 非补偿情况
ram=20 : HB0=(A-B)+(..)/(A+B)+(..) HB0 非补偿情况 综合
ram=21 : TMP=(p1-p2) / p2 温度
ram=22 : Status
ram=23 : HB1+ 在 SG_A1, 管脚 11 的时间测量 TDC
ram=24 : CAL TDC 分辨率
ram=25 : UBATT 测出的供电电压
ram=26-31 : NC 在 EEprom 编程时空闲
x-Akku : HB0 ram 值=20
y-Akku : Temp ram 值=21
z-Akku : Flags ram 值=63
ramadr : 0 RAM 地址指针的值

描述:
A : 在 SG_A1, Pin 11 的放电时间测量
B : 在 SG_A2, Pin 12 的放电时间测量

C： 在 SG_B1, Pin 8 的放电时间测量
D： 在 SG_B2, Pin 9 的放电时间测量
E： 在 SG_C1, Pin 5 的放电时间测量
F： 在 SG_C2, Pin 6 的放电时间测量
G： 在 SG_D1, Pin 2 的放电时间测量
H： 在 SG_D2, Pin 3 的放电时间测量
P1: 在 TMP_1, Pin 14 的放电时间测量
P2: 在 TMP_2, Pin 15 的放电时间测量

格式:
HB1: 结果单位为 100 ppm, HB1/100 =结果单位为 ppm
HB2: 结果单位为 100 ppm, HB2/100 = 结果单位为 ppm
HB3: 结果单位为 100 ppm, HB3/100 = 结果单位为 ppm
HB4: 结果单位为 100 ppm, HB4/100 = 结果单位为 ppm
HB0: 结果单位为 100 ppm, HB0/100 = 结果单位为 ppm
TMP: 结果(Tmp) = TMP/(1<<24)
Status: 见下面
HB1+: Result (HB1+)/ns = 250 * HB1+ /(1<<14) [4MHz 时钟]
CAL: Result (Cal)/ps = 250000 / CAL [4MHz 时钟]
UBATT: Result (UBATT)/V = 2.0+1.6*UBATT/64
HB1,HB2,HB3,HB4,HB0,TMP 是以 2 进制出现的。 MSB = 1 表明结果是一个负值。为了得到正结果请如下操作： h1000000000000-X.

3.4.3 状态寄存器

表 4

位	描述
Status[23]= flg_io3_mio	管脚 29
Status[22]= flg_io2_sck	管脚 22
Status[21]= flg_io1_sdi	管脚 21
Status[20]= flg_io0_sdo	管脚 20
Status[19]= flg_rstpwr	1 =上电复位使跳入 EEPROM
Status[18]= flg_rstssn	1 =按键按下使跳入 EEPROM
Status[17]= flg_wdtalt	1 =看门狗中断使跳入 EEPROM
Status[16]= flg_endavg	1 =测量结束使跳入 EEPROM
Status[15]= flg_intav0	1 =在休眠模式下跳入 EEPROM
Status[14]= flg_ub_low	1 =低电压
Status[13]= flg_errtdc	1 = TDC 错误
Status[12]= flg_pslock[1]	1 =相位移锁定
Status[11]= flg_pslock[0]	1 =相位移锁定
Status[10]= flg_errprt	1 =在应变测量口有错误
Status[09]= flg_timout	1 = TDC 溢出
Status[08]= 1'bx	1 = 不应用
Status[07]= flg_io3_mio_r	1 =在管脚 Pin29 的上升沿. = no edge
Status[06]= flg_io2_sck_r	1 =在管脚 Pin22 的上升沿, 0 = no edge
Status[05]= flg_io1_sdi_r	1 =在管脚 Pin21 的上升沿, 0 = no edge
Status[04]= flg_io0_sdo_r	1 =在管脚 Pin20 的上升沿, 0 = no edge
Status[03]= flg_io3_mio_f	1 =在管脚 Pin29 的下降沿, 0 = no edge
Status[02]= flg_io2_sck_f	1 =在管脚 Pin22 的下降沿, 0 = no edge
Status[01]= flg_io1_sdi_f	1 =在管脚 Pin21 的下降沿, 0 = no edge
Status[00]= flg_io0_sdo_f	1 =在管脚 Pin20 的下降沿, 0 = no edge

3.5 指令集

PS08 的完整指令集是由 94 条核心指令组成，是通过 CPU 解码的具有独特操作码的指令集。 另外有一个仿真指令， no2lcd, 可以自动由 PS08 汇编代替，然后调用在 ROM 代码的相应子程序。

指令集的多样性和综合性允许写入非常全面而且大量的程序而仅用 1 K EEPROM。

分支指令:

有 3 种在代码中转移的原则：
- 在整个地址空间内的 12 位的绝对地址转移。
- Branch. 相对于–128 to +127 位地址的 8 位实际地址。
- Skip. 跳过 3 个操作码 (3 到 15 bytes).

汇编将 jump 和 branch 放在一起构成新的操作码 goto。
只有当使用绝对 jump 的时候没有其它条件情况下才可以跳入子程序。

算术运算操作:
RAM 是 24 位字的结构。 所有的指令都是 2 进制为基础。一次算术操作命令结合了两个累加器然后将结果写入第一次提到过的累加器中。RAM 地址指针显示了 RAM 地址，这个地址的处理方式与累加器相同。每一次在累加器的操作都会影响到 4 个标志位。标志位以最后一次操作为准

表 5

简单算术	复杂算术	移动 & 旋转	RAM 控制
abs add compare compl decr getflag incr sign sub	div24 divmod mult24 mult	clrC rotl rotR setC shiftL shiftR	clear decramadr incramadr move pntramadr ramadr swap

逻辑运算	按位运算（Bitwise）	LCD 显示	EEPROM 控制
and eor nor invert nand nor or	bitclr nitinv bitset	dez2lcd newlcd no2lcd	equal getepr putepr

无条件转移	跳过标志	其它
goto gotoBitC gotoBitS gotoCarC gotoCarS gotoEQ gotoNE gotoNeg gotoOvrC gotoOvrS gotoPos jsub jsubret	skip skipBitC skipBitS skipCarC skipCarS skipEQ skipNE skipNeg skipOvrC skipOvrS skipPos	clk10kHz clrwdt initTDC newcyc nop stop

abs	**寄存器的绝对值**
句法:	abs p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := | p1 |
影响标志位:	C O S Z
字节:	2
循环:	2
描述:	寄存器值的绝对值
类别:	简单算术运算

add	**加法**
句法:	add p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或者 24-位数
算法:	p1 := p1 + p2
影响标志位:	C O S Z
字节:	2 (p2 = ACCU) 4 (p2 = number)
循环:	2 (p2 = ACCU) 4 (p2 = number)
描述:	两个寄存器相加或者一个常数和寄存器相加
类别:	简单算术运算

and	**逻辑与**
句法:	and p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或者 24-位
算法:	p1 := p1 AND p2
影响标志位:	S Z
字节:	2 (p2 = ACCU) 5 (p2 = number)
循环:	3 (p2 = ACCU) 6 (p2 = number)
描述:	两个寄存器的逻辑 AND 或者寄存器和常数的逻辑 AND
类别:	逻辑运算

bitclr	**清除单一位**
句法:	bitclr p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = number 0 到 23
算法:	p1:=p1 and not (1<<p2)
影响标志位:	S Z
字节:	2
循环:	2
描述:	在目标寄存器中清除其中一位
类别:	按位运算（Bitwise）

bitinv	**反相单一位**
句法:	bitinv p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = number 0 to 23
算法:	p1:=p1 eor (1<<p2)
影响标志位:	S Z
字节:	2

循环:	2
描述:	在目标寄存器中翻转其中一位
类别:	按位运算（Bitwise）

bitset	**设置单一位**
句法:	bitset p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = number 0 to 23
算法:	p1:=p1 or (1<<p2)
影响标志位:	S Z
字节:	2
循环:	2
描述:	在目标寄存器中设置其中一位
类别:	按位运算（Bitwise）

clear	**清除寄存器**
句法:	clear p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := 0
影响标志位:	S Z
字节:	1
循环:	1
描述:	将所被寻址的寄存器清 0
类别:	RAM 控制

clk10khz	**时钟源 10 kHz**
句法:	clk10khz p1
参数:	p1 = number 0 or 1
算法:	-
影响标志位:	-
字节:	2
循环:	3
描述:	将处理器的时钟源改变到 10 kHz. 处理器的时钟源被从 40 MHz 切换到低速 10kHz 时钟。10 kHz 时钟对于温度和供电电压变化很稳定。如果 p1 被设置为 1 那么 10 kHz 时钟则开启, 如果 p1 == 0 那么 10 kHz 时钟将被关闭。
类别:	其它

clrC	**清除标志位**
句法:	clrC
参数:	-
算法:	-
影响标志位:	C O
字节:	1
循环:	1
描述:	清除进位和溢出标志
类别:	移动和旋转

clrwdt	**清除看门狗**
句法:	clrwdt
参数:	-
算法:	-
影响标志位:	-
字节:	2
循环:	

描述:	清除看门狗. 这个操作码用于在程序运行结束后清出看门狗定时器。在'stop'之前应用此操作码.
类别:	其它

Compare	**比较两个数值**
句法:	compare p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或者 24-位数
算法:	---:=p2-p1 仅仅标志位被改变而不是寄存器
影响标志位:	C O S Z
字节:	1 (p1=ACCU, p2=ACCU) 4 (p1=ACCU, p2=NUMBER)
循环:	1 (p1=ACCU, p2=ACCU) 4 (p1=ACCU, p2=NUMBER)
描述:	通过相减比较 2 个寄存器 通过相减比较寄存器和一个常数 标志位根据相减的结果而改变，但是寄存器的内容本身不变
类别:	简单算术运算

compl	**2 进制补码**
句法:	compl p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := - p1 = not p1 + 1
影响标志位:	S Z
字节:	2
循环:	2
描述:	寄存器的 2 进制补码
类别:	简单算术运算

decr	**自减**
句法:	decr p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := p1 – 1
影响标志位:	C O S Z
字节:	1
循环:	1
描述:	寄存器自减
类别:	简单算术运算

decramadr	**RAM 地址指针自减**
句法:	decramadr
参数:	-
算法:	-
影响标志位:	-
字节:	1
循环:	1
描述:	将 ram 地址指针减少 1
类别:	Ram 操作

dez2lcd	**10 进制到段码转换**
句法:	dez2lcd p1
参数:	p1 = ACCU [x,y,z,r]
算法:	-
影响标志位:	-
字节:	2

循环:	3
描述:	转换在寄存器 p1 中的 10 进制代码到 7 段码。从寄存器 p1 的 0-9 的十进制数被转换到标准的 7 段码(数字 a-h). 这个操作码可以用于高级的 LCD 转换程序当中, 当 no2lcd 不够的时候 dez hgfe dcba 0 --> 0b00111111=0x3F 1 --> 0b00000110=0x06 2 --> 0b01011011=0x3B 3 --> 0b01001111=0x4F 4 --> 0b01100110=0x66 5 --> 0b01101101=0x6D 6 --> 0b01111101=0x7D 7 --> 0b00000111=0x07 8 --> 0b01111111=0x7F 9 --> 0b01101111=0x6F
Category:	LCD 显示

div24	**带符号除法 24 位**
句法:	div24 p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r]
算法:	p1 := (p1 << 24) / p2 (if |p1| < 2 * |p2|)
影响标志位:	S & Z of p1
字节:	2
循环:	20
描述:	两个寄存器的带符号除法 24 位的两个寄存器除法，结果被分配到 p1
类别:	复杂算术运算

divmod	**带符号模除法**
句法:	divmod p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r]
算法:	p1 := p1 / p2 和 p2 := p1 % p2
影响标志位:	S Z
字节:	2
循环:	
描述:	两个寄存器的带符号模除法运算 两个寄存器 24 高位除法，结果被分配到 p1。余数被分配到 p2。
类别:	复杂算术运算

eor	**异或**
句法:	eor p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或 24-位数
算法:	p1 := p1 xor p2 按位异或 0 / 0 和 1 / 1 返回 0 按位异或 0 / 1 和 1 / 0 返回 1
影响标志位:	S Z
字节:	2 (p1=ACCU, p2=ACCU) 5 (p1=ACCU, p2=NUMBER)
循环:	3 (p1=ACCU, p2=ACCU) 6 (p1=ACCU, p2=NUMBER)
描述:	两个给定寄存器的逻辑 XOR (eXclusive OR, 不相等) 常数和寄存器的逻辑 XOR (eXclusive OR, 不相等)
类别:	逻辑运算

eorn	**异或非**
句法:	eorn p1,p2

参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或 24-位数
算法:	p1 := p1 xnor p2 按位运算 0 / 0 和 1 / 1 返回 1 按位运算 0 / 1 和 1 / 0 返回 0
影响标志位:	S Z
字节:	2 (p1=ACCU, p2=ACCU) 5 (p1=ACCU, p2=NUMBER)
循环:	3 (p1=ACCU, p2=ACCU) 6 (p1=ACCU, p2=NUMBER)
描述:	两个给定寄存器的逻辑 XNOR (eXclusive NOR, 相等) 常数和寄存器的逻辑 XNOR (eXclusive NOR, 相等)
类别:	逻辑运算

equal	**写 3 Bytes 到 EEPROM**
句法:	equal p1
参数:	p1 = 24-Bit number
算法:	-
影响标志位:	-
字节:	3
循环:	
描述:	写 3bytes 到 (p1) EEPROM 的配置寄存器。 Equal 操作码被用来直接写 3 个 bytes 到 EEPROM 寄存器的配置数据当中。因此这个操作码在汇编程序开始时将被直接应用 16 次通过 p1 给配置数据，比如像 'putepr'这个操作码。EEPROM 的配置是在从 0..47byte 空间中完成，与 16 x 24 位的寄存器结合。从 48byte 开始，用户可以写自己的代码到 EEPROM。应用这个操作码，可以写如您自己的配置寄存器代替标准的配置。
类别:	EEPROM 操作

getepr	**获得 EEPROM 内容**
句法:	getepr p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := EEPROM 寄存器内容(通过 ram 地址指针寻址)
影响标志位:	S Z
字节:	1
循环:	6
描述:	将 EEPROM 内容给到寄存器当中。被指配的结果寄存器 p1 通过 ram 地址指针的操作获得 EEPROM 寄存器的内容。这个操作码需要在程序计数堆栈中一个临时的空间(详细说明见下面)。
类别:	EEprom 操作

getflag	**置位 S 和 Z 标志**
句法:	getflag p1
参数:	p1 = ACCU [x,y,z,r]
算法:	signum := set if p1 < 0 notequalzero := set if p1 <> 0
影响标志位:	S Z
字节:	1
循环:	1
描述:	置位符号和非零标志位根据被寻址的寄存器，寄存器的内容不受影响。
类别:	简单算数运算

goto	**无条件跳转**
句法:	goto p1
参数:	p1 = JUMPLABEL
算法:	PC:= p1
影响标志位:	-

字节:	2 [相对跳转] 3 [绝对跳转]
循环:	3 [相对跳转] 4 [绝对跳转]
描述:	无条件跳转。程序计数器被设置到目标地址上。 目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	无条件跳转

gotoBitC	**位清零跳转**
句法:	gotoBitC p1, p2, p3
参数:	p1 = ACCU [x,y,z,r] p2 = NUMBER [0..23] p3 = JUMPLABEL
算法:	if [寄存器 p1 的位 p2 == 0] PC := p3
影响标志位:	-
字节:	3
循环:	4
描述:	位清零跳转。程序计数器将会被设置到目标地址如果在寄存器 p1 的所选择位被清零的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	位运算

gotoBitS	**当所选位被置时跳转**
句法:	gotoBitS p1, p2, p3
参数:	p1 = ACCU [x,y,z,r] p2 = NUMBER [0..23] p3 = JUMPLABEL
算法:	if [寄存器 p1 的位 p2 == 1] PC := p3
影响标志位:	-
字节:	3
循环:	4
描述:	当所选位被置时跳转。程序计数器将会被设置到目标地址如果在寄存器 p1 的所选择位被置位的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	位运算

gotoCarC	**进位清零跳转**
句法:	gotoCarC p1
参数:	p1 = JUMPLABEL
算法:	if [carry == 0] PC := p1
影响标志位:	-
字节:	2 [相对跳转] 3 [绝对跳转]
循环:	3 [相对跳转] 4 [绝对跳转]
描述:	当进位清零时跳转。 。程序计数器将会被设置到目标地址当进位标志清零的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoCarS	**进位被置位时跳转**
句法:	gotoCarS p1
参数:	p1 = JUMPLABEL
算法:	if [carry == 1] PC := p1

影响标志位:	-
字节:	2 (相对跳转) 3 (绝对跳转)
循环:	3 (相对跳转) 4 (绝对跳转)
描述:	进位被置位时跳转。程序计数器将会被设置到目标地址当进位标志被置位的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoEQ	**等于零跳转**
句法:	gotoEQ p1
参数:	p1 = JUMPLABEL
算法:	if (Z == 0) PC := p1
影响标志位:	-
字节:	2 (相对跳转) 3 (绝对跳转)
循环:	3 (相对跳转) 4 (绝对跳转)
描述:	当等于零的时候跳转。程序计数器将会被设置到目标地址当前面的结果为零的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoNE	**非零时跳转**
句法:	gotoNE p1
参数:	p1 = JUMPLABEL
算法:	if (Z == 1) PC := p1
影响标志位:	-
字节:	2 (相对跳转) 3 (绝对跳转)
循环:	3 (相对跳转) 4 (绝对跳转)
描述:	当结果不等于零时跳转。程序计数器将会被设置到目标地址当前面的结果为不等于零的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoNeg	**负数值跳转**
句法:	gotoNeg p1
参数:	p1 = JUMPLABEL
算法:	if (S == 1) PC := p1
影响标志位:	-
字节:	2 (相对跳转) 3 (绝对跳转)
循环:	3 (相对跳转) 4 (绝对跳转)
描述:	结果为负值时跳转。程序计数器将会被设置到目标地址当前面的结果为负值的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoOvrC	**溢出标志清零跳转**
句法:	gotoOvrC p1
参数:	p1 = JUMPLABEL
算法:	if (O == 0) PC := p1
影响标志位:	-
字节:	2 (相对跳转)

	3 [绝对跳转]
循环:	3 [相对跳转] 4 [绝对跳转]
描述:	当溢出标志位清零的时候跳转。程序计数器将会被设置到目标地址当溢出标志清零的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoOvrS	**溢出标志位置位跳转**
句法:	gotoOvrS p1
参数:	p1 = JUMPLABEL
算法:	if (O == 1) PC := p1
影响标志位:	-
字节:	2 [相对跳转] 3 [绝对跳转]
循环:	3 [相对跳转] 4 [绝对跳转]
描述:	当溢出标志位置位的时候跳转。程序计数器将会被设置到目标地址当溢出标志被置位的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

gotoPos	**正值跳转**
句法:	gotoPos p1
参数:	p1 = JUMPLABEL
算法:	if (S == 0) PC := p1
影响标志位:	-
字节:	2 [相对跳转] 3 [绝对跳转]
循环:	3 [相对跳转] 4 [绝对跳转]
描述:	当结果为正值的时候跳转。程序计数器将会被设置到目标地址当结果为正数的时候。目标地址通过应用一个 jumplabel 给出。参见例程部分来了解如何引入一个 jumplabel。
类别:	Goto on flag

incr	**自增**
句法:	incr p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := p1 + 1
影响标志位:	C O S Z
字节:	1
循环:	1
描述:	寄存器增加 1
类别:	简单算术运算

incramadr	**RAM 地址自增**
句法:	incramadr
参数:	-
算法:	-
影响标志位:	-
字节:	1
循环:	1
描述:	RAM 地址指针加 1
类别:	RAM 操作

initTDC	**初始化 TDC**
句法:	initTDC
参数:	-
算法:	-
影响标志位:	-
字节:	2
循环:	3
描述:	TDC (time-to-digital converter)的初始化操作。应该在配置寄存器结束后发送。这个初始化操作将不会改变寄存器的配置。
类别:	其它

invert	**按位翻转**
句法:	invert p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := not p1
影响标志位:	S Z
字节:	2
循环:	2
描述:	对寄存器按位翻转
类别:	逻辑运算

jsub	**无条件跳转**
句法:	jsub p1
参数:	p1 = JUMPLABEL
算法:	PC := p1
影响标志位:	C O S Z
字节:	3
循环:	4
描述:	无条件跳转到子程序中。程序计数器通过 jumplabel 给出的地址所增加。子程序一直运行直到 关键字 'jsubret' 出现。然后将会跳回来执行 jsub-call 下面的命令。这个操作码在程序计数堆栈中需要一个临时的空间 (解释见下面)。
类别:	无条件跳转

jsubret	**从子程序中返回**
句法:	jsubret
参数:	-
算法:	PC := PC from jsub-call
影响标志位:	-
字节:	1
循环:	3
描述:	从子程序中返回。子程序可以通过呼叫 'jsub'调用然后通过呼叫 jsubret 来退出。程序将继续执行 jsub-call 后面的命令。必须通过 jsubret 关闭子程序 – 否则将不会跳回。
类别:	无条件跳转

move	**移动**
句法:	move p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或者 24-常数
算法:	p1 := p2
影响标志位:	S Z
字节:	1 (p1=ACCU, p2=ACCU) 4 (p1=ACCU, p2=NUMBER)
循环:	1 (p1=ACCU, p2=ACCU) 4 (p1=ACCU, p2=NUMBER)

描述:	将 p2 的内容移动到 p1 (p1=ACCU, p2=ACCU) 将常数移动到 p1 (p1=ACCU, p2=NUMBER)
类别:	RAM access

mult24	**带符号的 24 位乘法**
句法:	mult24 p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r]
算法:	p1 := (p1 * p2) >> 24
影响标志位:	S & Z of p1
字节:	2
循环:	30
描述:	带符号的两个寄存器的乘法运算比如 mult48。但是只有两个寄存器的高 24 位相乘，结果存在 p1 当中。
类别:	复杂算术运算

mult	**带符号的 48 位乘法**
句法:	mult48 p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r]
算法:	p1,p2 := p1 * p2
影响标志位:	S & Z of p1
字节:	2
循环:	30
描述:	带符号的两个寄存器的乘法运算 高 24 位结果存储到 p1 低 24 位结果存储到 p2
类别:	复杂算术运算

nand	**逻辑与非**
句法:	nand p1,p2
参数:	p1 = ACCU [x,y,z,r] p1 = ACCU [x,y,z,r] 或者 24-位常数
算法:	p1 := p1 nand p2 仅在相应位为 1 / 1 的时候返回 0
影响标志位:	S Z
字节:	2 (p1=ACCU, p2=ACCU) 5 (p1=ACCU, p2=NUMBER)
循环:	3 (p1=ACCU, p2=ACCU) 6 (p1=ACCU, p2=NUMBER)
描述:	两个给定寄存器的逻辑 NAND (反向 AND) 寄存器和常数的逻辑 NAND (反向 AND)
类别:	逻辑运算

newcyc	**Start TDC**
句法:	newcyc
参数:	-
算法:	-
影响标志位:	-
字节:	2
循环:	3
描述:	TDC 测量开始。 这个操作码在 PS08 配置寄存器和初始化结束后开始新的测量循环。一般这是通过 PS08 的 rom 程序本身完成，但是当用户指定的复位被应用得时候，这个操作码就有一定的用途了。
类别:	其它

newlcd	**载入新的 LCD 数据**

句法:	newlcd
参数:	-
算法:	-
影响标志位:	-
字节:	2
循环:	3
描述:	载入新的 LCD 数据。新的段码数据从寄存器 61-63 写入 LCD 驱动。 刷新显示器.
类别:	LCD 显示

no2lcd	**转换 6 个数字到 LCD 码**
句法:	no2lcd
参数:	-
算法:	-
影响标志位:	-
字节:	3
循环:	呼叫子程序
描述:	将寄存器 x 中的 6 位十进制数转换为 7-段码。 小数点的位置是由寄存器 y 中的 T 数 (0..5)来决定的。小数点前面的零将会被清除。 转换的结果直接写到 LCD 寄存器 61-62 与 'newlcd'一同应用。 实际上这个操作码是在调用 ROM 中的一个子程序。PS08 汇编将这个命令转换成相应的跳转命令。
类别:	LCD 显示

nop	**无操作**
句法:	-
参数:	-
算法:	-
影响标志位:	-
字节:	1
循环:	1
描述:	空间占用码或者时间调整(没有功能)
类别:	其它

nor	**逻辑 NOR**
句法:	nor p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或 24-位常数
算法:	p1 := p1 nor p2 仅在相应位为 0 / 0 时返回 1
影响标志位:	S Z
字节:	2 (p1=ACCU, p2=ACCU) 5 (p1=ACCU, p2=NUMBER)
循环:	3 (p1=ACCU, p2=ACCU) 6 (p1=ACCU, p2=NUMBER)
描述:	两个寄存器的逻辑 NOR 运算 (反向 OR) 寄存器和常数的逻辑 NOR 运算(反向 OR)
类别:	Logic

or	**逻辑 OR**
句法:	or p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r] 或 24-位常数
算法:	p1 := p1 or p2 仅当相对应位为 0 / 0 时返回 0
影响标志位:	S Z

字节:	2 (p1=ACCU, p2=ACCU) 5 (p1=ACCU, p2=NUMBER)
循环:	3 (p1=ACCU, p2=ACCU) 6 (p1=ACCU, p2=NUMBER)
描述:	两个寄存器的逻辑 OR 运算 寄存器和常数的逻辑 OR 运算
类别:	逻辑运算

pntramadr	**选择 RAM 地址指针**
句法:	pntramadr p1
参数:	p1 = [0,1]
算法:	-
影响标志位:	-
字节:	1
循环:	1
描述:	选择 ram 地址指针 0 = ramaddress pointer 1 1 = ramaddress pointer 2 可能会用在子程序当中
类别:	RAM 操作

putepr	**将寄存器内容放入 EEPROM**
句法:	putepr p1
参数:	p1 = ACCU [x,y,z,r]
算法:	EEPROM 寄存器 (通过 ram 地址指针寻址) := p1
影响标志位:	-
字节:	4
循环:	65536 = ~50ms
描述:	将寄存器内容放入 EEPROM。 将所指定地址的寄存器 p1 移动到 EEPROM (EEPROM 寄存器地址通过 ram 地址指针来设置)。 仅 EEPROM 寄存器 14 和 15 可以通过 'putepr' 来进入。这个操作码在程序计数堆栈中需要一个临时空间 (解释见下面). 'putepr' 不可以与 skip-操作码同时使用因为这个操作码的执行时间太长 (大约: 50ms)
类别:	EEPROM 操作

ramadr	**设置 RAM 地址指针**
句法:	ramadr p1
参数:	p1 = 5-位常数
算法:	-
影响标志位:	-
字节:	1
循环:	1
描述:	设置指针到 ram 地址 (范围: 0..65)
类别:	RAM 操作

rotL	**向左旋转**
句法:	rotL p1[,p2]
参数:	p1 = ACCU [x,y,z,r] p2 = 4-位常数或无
算法:	p1 := p1<< 1+ carry; carry:=MSB(x) (in case rotL p1, without p2) p1 := repeat (p2) rotL p1 (in case rotL p1,p2)
影响标志位:	C O S Z (of the last step)
字节:	1 (p1=ACCU, p2=none) 2 (p1=ACCU, p2=NUMBER)
循环:	1 (p1=ACCU, p2=none) 1+p2 (p1=ACCU, p2=NUMBER)

描述:	向左旋转 p1 --> 向左移动 p1 寄存器,将进位寄存器的位给 LSB, MSB 被放在进位寄存器中。 将 p1 寄存器带有进位向左旋转 p2 次 -->将 p1 寄存器带有进位向左旋转 p2 次, 在每次旋转中将进位放到 LSB 中，将 MSB 位放到进位中。
类别:	移动 & 旋转

rotR	**向右旋转**
句法:	rotR p1{,p2}
参数:	p1 = ACCU [x,y,z,r] p2 = 4-位常数或无
算法:	p1 := p1>> 1+ carry; carry: =MSB(x) (in case rotR p1, without p2) p1 := repeat (p2) rotR p1 (in case rotR p1,p2)
影响标志位:	C O S Z (of the last step)
字节:	1 (p1=ACCU, p2=none) 2 (p1=ACCU, p2=NUMBER)
循环:	1 (p1=ACCU, p2=none) 1+p2 (p1=ACCU, p2=NUMBER)
描述:	向右旋转 p1 -->向右移动 p1 寄存器,将进位寄存器的位给 MSB, LSB 被放在进位寄存器中。 将 p1 寄存器带有进位向右旋转 p2 次 -->将 p1 寄存器带有进位向右旋转 p2 次, 在每次旋转中将进位放到 MSB 位中，将 LSB 位放到进位中。
类别:	移动 & 旋转

setC	**置位进位标志**
句法:	setC
参数:	-
算法:	-
影响标志位:	C O
字节:	1
循环:	1
描述:	置位进位标志和清除溢出标志
类别:	**移动 & 旋转**

shiftL	**向左移位**
Syntax:	shiftL p1,(p2)
句法:	p1 = ACCU [x,y,z,r] p2 = 4-位常数或无
参数:	p1 := p1<< 1; carry :=MSB(x) (in case rotL p1, without p2) p1 := repeat (p2) shiftL p1 (in case rotL p1,p2)
算法:	C O S Z
影响标志位:	1 (p1=ACCU, p2=none) 2 (p1=ACCU, p2=NUMBER)
字节:	1 (p1=ACCU, p2=none) 1 + p2 (p1=ACCU, p2=NUMBER)
循环:	将 p1 左移 --> 将 p1 寄存器的内容左移, 将 0 放入 LSB 位, 将 MSB 位放入进位中。 将 p1 左移 p2 次 --> 将 p1 寄存器内容左移 p2 次, 在每次移动中将 0 放入 LSB 位, 将 MSB 位放入进位中。
描述:	**移动 & 旋转**

shiftR	**向右移位**
句法:	shiftR p1,(p2)
参数:	p1 = ACCU [x,y,z,r]

	p2 = 4-位常数或无
算法:	p1 := p1>> 1; carry:=MSB(x) (in case rotL p1, without p2) p1 := repeat (p2) shiftL p1 (in case rotL p1,p2)
影响标志位:	C O S Z
字节:	1 (p1=ACCU, p2=none) 2 (p1=ACCU, p2=NUMBER)
循环:	1 (p1=ACCU, p2=none) 1 + p2 (p1=ACCU, p2=NUMBER)
描述:	带符号右移寄存器 p1 --> 右移寄存器 p1 内容, MSB 位根据数值是正数还是负数被复制保留。 带符号的将 p1 右移 p2 次-->将 p1 寄存器内容右移 p2 次, MSB 位根据数值是正数还是负数被复制保留。
类别:	**移动 & 旋转**

sign	**符号**
句法:	sign p1
参数:	p1 = ACCU [x,y,z,r]
算法:	p1 := p1 / | p1 | p1 := 1 = 0x000001 if p1 >= 0 p1 := -1 = 0xFFFFFF if p1 < 0
影响标志位:	S Z
字节:	2
循环:	2
描述:	以 2 的补码形式的带符号的被指定的寄存器。 一个正数返回 1， 一个负数返回 -1 零被考虑为正数
类别:	简单算术运算

skip	**跳过**
句法:	skip p1
参数:	p1 = NUMBER [1,2,3]
算法:	PC := PC + bytes of next p1 lines
影响标志位:	
字节:	1
循环:	1 + 跳过的命令
描述:	无条件跳过 p1
类别:	无条件跳转

skipBitC	**有条件跳过**
句法:	skipBitC p1,p2,p3
参数:	p1 = ACCU [x,y,z,r] p2 = NUMBER[0..23] p2 = NUMBER[1,2,3]
算法:	if (寄存器 p1 的 p2 位 == 0) PC := PC + bytes of next p3 lines
影响标志位:	-
字节:	1
循环:	1 + 跳过的命令
描述:	如果寄存器 p1 的 p2 位被清零则跳过 p3 命令
类别:	按位运算

skipBitS	**有条件跳过**
句法:	skipBitS p1,p2,p3
参数:	p1 = ACCU [x,y,z,r] p2 = NUMBER[0..23]

	p2 = NUMBER[1,2,3]
算法:	if (寄存器 p1 的 p2 位== 1) PC := PC + bytes of next p3 lines
影响标志位:	-
字节:	1
循环:	1 +跳过的命令
描述:	当寄存器 p1 的 p2 位被置位时跳过 p3 命令
类别:	按位运算

skipCarC	**进位清零跳过**
句法:	skipCarC p1
参数:	p1 = NUMBER [1,2,3]
算法:	if (carry == 0) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 +跳过的命令
描述:	如果进位被清零跳过 p1 命令
类别:	跳过标志

skipCarS	**进位置位跳过**
句法:	skipCarS p1
参数:	p1 = NUMBER [1,2,3]
算法:	if (carry == 1) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 +跳过的命令
描述:	当进位标志被置位时跳过 p1 命令
类别:	跳过标志

skipEQ	**等于零跳过**
句法:	skipEQ p1
参数:	p1 = NUMBER[1,2,3]
算法:	if (notequalzero == 0) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 + 跳过的命令
描述:	如果前面操作的结果等于零时跳过 p1 命令
类别:	跳过标志

skipNE	**非零跳过**
句法:	skipNE p1
参数:	p1 = NUMBER[1,2,3]
算法:	if (notequalzero == 1) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 + 跳过的命令
描述:	如果前面操作的结果不等于零时跳过 p1 命令

类别:	跳过标志

skipNeg	**负值跳过**
句法:	skipNeg p1
参数:	p1 = NUMBER[1,2,3]
算法:	if (signum == 1) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 + 跳过的命令
描述:	当上一次操作的结果小于零时跳过 p1 命令
类别:	跳过标志

skipOvrC	**溢出清零跳过**
句法:	skipOvrC p1
参数:	p1 = NUMBER[1,2,3]
算法:	if (overflow == 0) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 +跳过的命令
描述:	当溢出标志位清零时跳过 p1 命令
类别:	跳过标志

skipOvrS	**溢出跳过**
句法:	skipOvrS p1
参数:	p1 = NUMBER[1,2,3]
算法:	if (overflow == 1) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 +跳过的命令
描述:	跳过 p1 命令如果溢出位被置
类别:	跳过标志

skipPos	**正值跳过**
句法:	skipPos p1
参数:	p1 = NUMBER[1,2,3]
算法:	if (signum == 0) PC := PC + bytes of next p1 lines
影响标志位:	-
字节:	1
循环:	1 + 跳过的命令
描述:	跳过 p1 命令如果前面的操作结果大于等于 0
类别:	跳过标志

stop	**Stop**
句法:	stop
参数:	-
算法:	-
影响标志位:	-
字节:	1
循环:	1

描述:	停止时间数字转换器的工作。 时钟产生源被停止，转换器和 EEPROM 进入休眠状态。对于转换的在启动可以通过外部事件如'看门狗定时器'，'外部开关'或者'新的金属应变测量结果'来触发。一般这个操作码是汇编程序上面的最后一句命令。
类别:	其它

sub	**减法**
句法:	sub p1,p2
参数:	p1 = NUMBER[1,2,3] p2 = NUMBER[1,2,3] or 24-位常数
算法:	p1:= p2 – p1
影响标志位:	C O S Z
字节:	1 (p1=ACCU, p2=ACCU) 4 (p1=ACCU, p2=NUMBER)
循环:	1 (p1=ACCU, p2=ACCU) 4 (p1=ACCU, p2=NUMBER)
描述:	2 个寄存器的减法 一个寄存器和一个常数的减法
类别:	简单算术运算

swap	**交换**
句法:	swap p1,p2
参数:	p1 = ACCU [x,y,z,r] p2 = ACCU [x,y,z,r]
算法:	p1 := p2 and p2 := p1
影响标志位:	-
字节:	1
循环:	3
描述:	交换两个寄存器内容 两个寄存器的内容被交换
类别:	RAM 操作

4 系统 Reset, 休眠模式和自动配置

在一次 Reset 或 TDC (测量结束后)是需要开启 ALU（算术逻辑运算单元）的。Reset 请求要比 TDC 有优先权。ALU 在被触发之后跳入从地址 1024 开始的代码中。
无论 ALU 是否在 Reset 后面开启都会进行第一次检测。

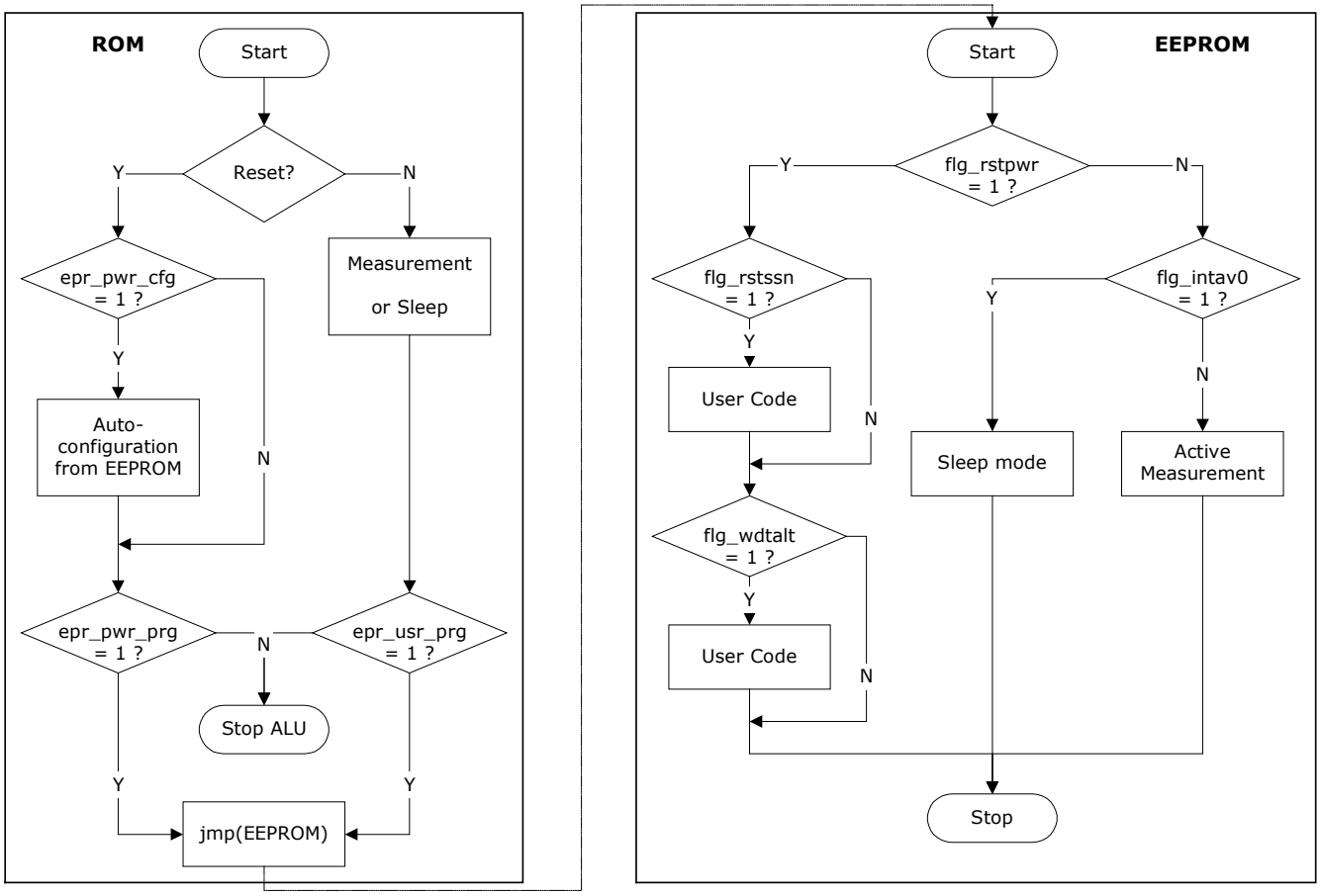
在一次 Reset 的情况下标志位 epr_pwr_cfg 被检测用来决定是否需要将 EEPROM 中的自动配置数据拷贝到 RAM 中。

在下面的标志位 epr_pwr_prg 被检查来决定是否 EEPROM 代码(从地址 48 开始)需要被执行。如果是，那么程序将跳入 EEPROM，否则微处理器将会停止。

如果 ALU 在休眠模式下被开启而不是通过 reset 开启的话，TDC 测量单元会启动一次测量。之后标志位 epr_usr_prg 被查看来决定是否跳入 EEPROM 中 (地址 48).

在 EEPROM 代码中首先要检查 flg_rstpwr 位来看一下跳转的原因是 reset 还是其它。如果是 reset 的话，那么进一步的原因检测需要通过后面的用户代码来完成。
否则对标志 flg_intav0 的检测将表明一次活动的应变测量。

13



在最后 ALU 将停止测量。这将给 ALU 一个完全的 reset 包括开始的标志位。程序硬件堆栈也将复位。只有在 RAM 中的数据保持不变。

4.1 上电复位

当在芯片上加电源电压的时候一次上电复位将会产生。整个芯片将会被重置，仅 RAM 保持不变。

在位 epr_pwr_prg 被设置为 1 的时候 (位 1, 寄存器 1)将从地址线 48 开始 EEPROM 程序代码。

4.2 看门狗复位

一次上电复位也可以由看门狗定时器触发。这种情况是在微处理器被 4 次启动而没有通过操作码"clrwdt"复位的时候发生。寄存器 22 的状态标志位 flg_wdtalt 中的位 17,表示了看门狗定时器的溢出。
在位 epr_pwr_prg 被设置为 1 的时候 (bit1, 寄存器 1)将从地址线 48 的 EEPROM 程序代码开始。

4.3 在管脚 27 的外部复位

在 stand-alone 模式下 (SPI_ENA = 0)可以通过管脚 27 (SPI_CSN_RST)给芯片一个外部的上电操作。 这可以作为一个外部复位按钮。 按钮的状态可以通过寄存器 22 中位 18 这个状态位 flg_rstssn 来确定。

如果位 epr_pwr_prg 被设置为 1 (bit1, 寄存器 1) EEPROM 程序代码将从地址 48 开始。

4.4 休眠模式

在休眠模式仅 10 kHz 晶振电路运行。在规律的时间间隔下控制器被唤醒但是并不做任何测量。在这种情况下可以检查 I/O's 端口。
微处理器从休眠模式的启动将从寄存器 22，位 22 状态位 flg_intav0 表示出来。

配置:
tdc_sleepmode register 1,位 17
tdc_conv_cnt[11:0] register0，位 23-14

休眠模式是通过设置 tdc_sleepmode 为 1 来唤醒的。这相当于设置 avrate=0。
在休眠模式下当 tdc_conv_cnt 转换计数器运行结束，从 Eeprom 的 48 位开始用户代码将马上开始运行。
初始化:
TDC 必须要在编写休眠模式之后初始化。

5 CPU 时钟产生

系统的基础时钟为内部低电流 10 kHz 振荡器。 它是用来在单一转换模式中触发测量用的。它的作用为：

- 在单一转换模式中触发测量
- 测量范围 2 中作为测量的预计数器
- 在 streched 模式下作为 cycletime 的基本时间

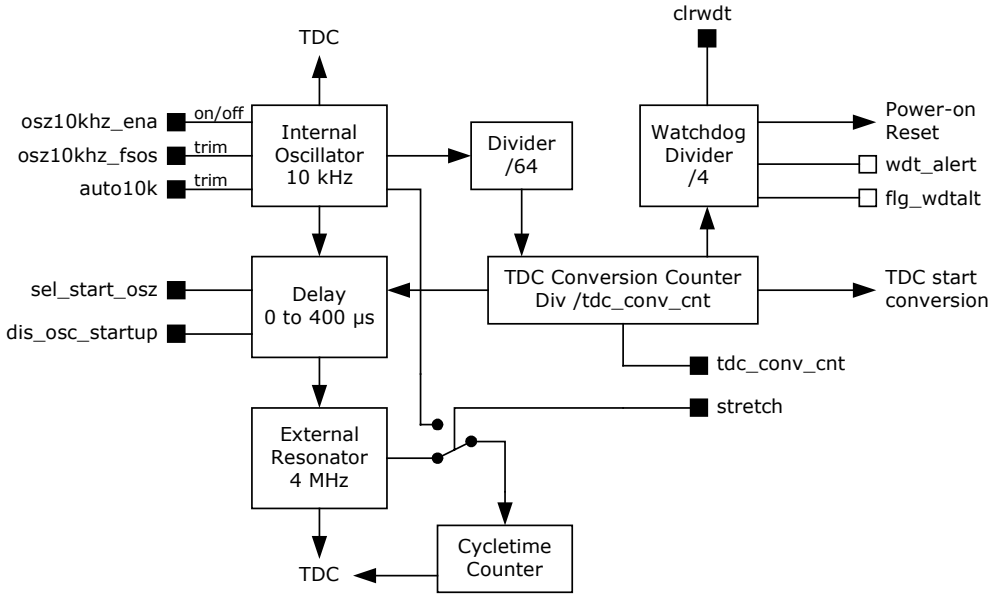


图 14

5.1 看门狗计数器和单一转换计数器

单一转换模式计数器在单一转换模式下触发一次测量。它是持续运行的。只有当设置位 osz10khz_ena 的时候这个计数器才会停止。单一转换的频率是由 10kHz / 64 / tdc_conv_cnt 给出。

在一次测量开始的时候看门狗计数器将会增加。看门狗记录了转换的次数。 在一次测量之后单片机开始运行用户代码。在普通操作模式通过 CLRWDT 的看门狗必须在用户代码结束前复位。如果 TDC 由于测量错误或者 EEPROM 代码不工作而导致的测量没有完成时看门狗将会引起一次上电复位。

在对 PS08 进行控制的时候可以通过 SPI 接口发送操作码(SPI_ENA = 1)watch_dog_off 来关闭看门狗。另外看门狗还可以通过在 SPI_CSN_RST 管脚的信号边沿进行复位。

6 IO-管脚

PS08 有 5 个 I/O 管脚: SPI_DO_IO0, SPI_DI_IO1, SPI_CLK_IO2, SEL_WHEAT_IO3 和 SPI_CSN_RST。

当芯片工作在 stand-alone 模式下的时候 (SPI 接口不可用 SPI_ENA=1)管脚 SPI_DO_IO0, SPI_DI_IO1, SPI_CLK_IO2, SEL_WHEAT_IO3 和 SPI_CSN_RST 被编程为带有上拉和下拉电阻的输入或输出管脚。当惠斯通电桥没有被使用的时候管脚 SEL_WHEAT_IO3 仅可以被用作输入管脚。

当 SPI 接口不用的时候 (SPI_ENA = 0)管脚 27, SPI_CSN_RST 可以被用作复位输入。这个时候输入需要一个外部的下拉电阻接到地，复位为高平有效。

6.1 配置管脚

管脚 29	SEL_WHEAT_IO3	Configreg_11, bit 22,23	io_en_3_mio
管脚 22	SPI_CLK_IO2	Configreg_11, bit 20,21	io_en_3_sck
管脚 21	SPI_SDI_IO1	Configreg_11, bit 18,19	io_en_3_sdi
管脚 20	SPI_SDO_IO0	Configreg_11, bit 16,17	io_en_3_sdo
端口定义	00 = 输入		
	01 = 带下拉的输入		
	10 = 带上拉的输入		
	11 = 输出		

6.2 输出-写

在配置寄存器 1 中设置输出。

管脚 29	SEL_WHEAT_IO3	Configreg_01, bit 13	io_a[3]
管脚 22	SPI_CLK_IO2	Configreg_01, bit 12	io_a[2]
管脚 21	SPI_SDI_IO1	Configreg_01, bit 11	io_a[1]
管脚 20	SPI_SDO_IO0	Configreg_01, bit 10	io_a[0]

6.3 输入-读

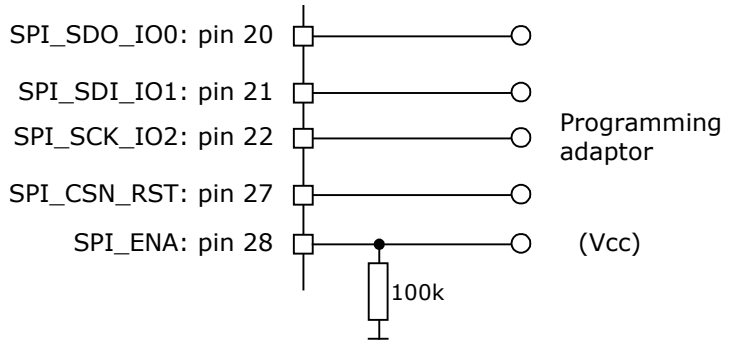
Status[23]= flg_io3_mio	管脚 29
Status[22]= flg_io2_sck	管脚 22
Status[21]= flg_io1_sdi	管脚 21
Status[20]= flg_io0_sdo	管脚 n20
Status[07]= flg_io3_mio_r	在管脚 29 的上升沿
Status[06]= flg_io2_sck_r	在管脚 22 的上升沿
Status[05]= flg_io1_sdi_r	在管脚 21 的上升沿
Status[04]= flg_io0_sdo_r	在管脚 20 的上升沿
Status[03]= flg_io3_mio_f	在管脚 29 的下降沿
Status[02]= flg_io2_sck_f	在管脚 22 的下降沿
Status[01]= flg_io1_sdi_f	在管脚 21 的下降沿
Status[00]= flg_io0_sdo_f	在管脚 20 的下降沿

7 SPI-接口

7.1 I 接口

SPI 接口用于写入 EEPROM 里面程序，配置数据，校准数据。
它还可以用于当 ps08 作为纯的转换器时与外部单片机进行通信用。在这个情况下下拉电阻将不必使用。
将 SPI_ENA 置高将会开启 SPI 接口，那么所属管脚将作为 SPI 接口的通信管脚不再作为普通 I/O 口用。在每次操作码发送之前必须要发送一个正脉冲给 CSN 线。

图 15



7.2 SPI 时序

这里我们仅描述了 PS08 作为纯时间数字转换器时的 SPI 与外部单片机通信的操作。

PS08 仅支持 4 个可能的通信模式中的一个模式:

时钟相位 = 1,时钟极性 = 0

数据是在时钟的下降沿被传输的。时钟是从低平开始。

图 16

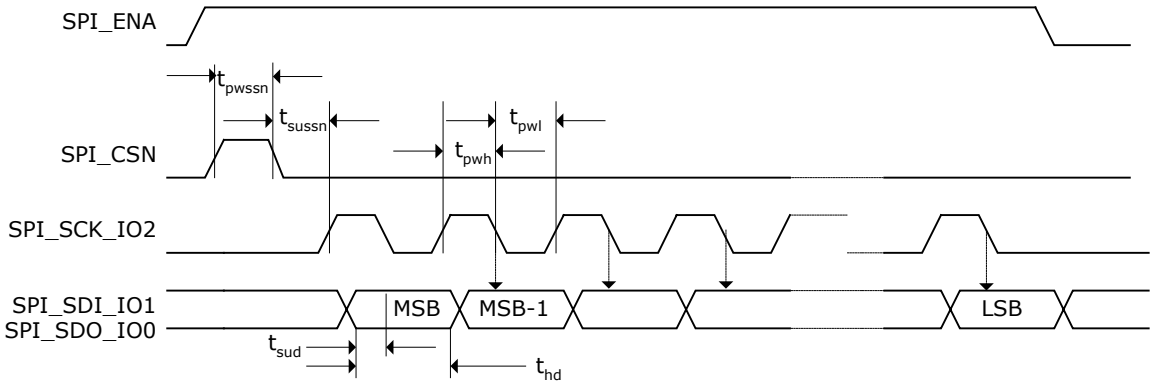


表 6 SPI 时序

时间:	**描述:**	**t_{min} [ns]**
tpwssn	SSN 脉冲宽度	500
tsussn	SSN / SCK 准备时间	500
tpwh	SCK 高平脉冲宽度	500
tpwl	SCK 低平脉冲宽度	500
tsud	数据准备时间	30
thd	数据保持时间	30

tpwh 和 tpwl 一起定义了 SPI 接口的时钟频率。那么 1µs 相当于 SPI 以 1MHz 速度传输。

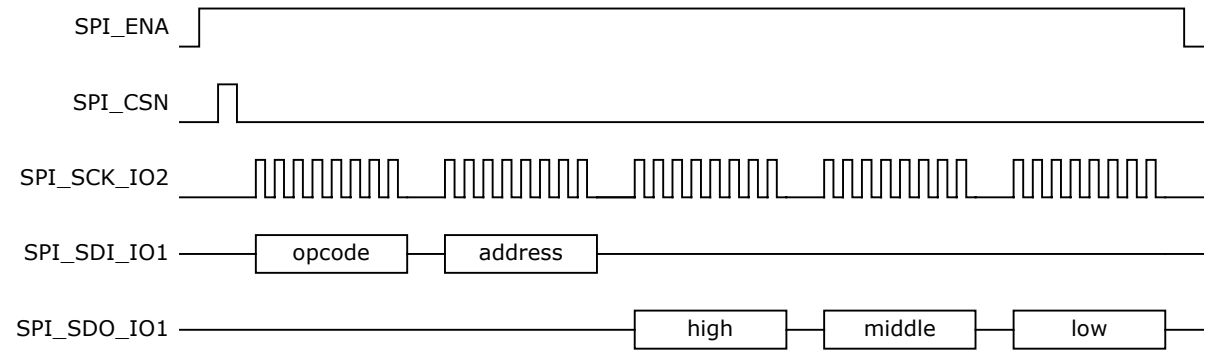
7.3 SPI-指令

RAM Write	= b00000000	= h00	
RAM Read	= b01000000	= h40	
New_LCD	= b01000110	= h46	
Power reset	= b11110000	= hF0	
Init reset	= b11000000	= hC0	
Start_new_cycle	= b11001100	= hCC	(持续工作模式)
Start_TDC_cycle	= b11001110	= hCE	(单一转换模式)
watch_dog_off	= b10011110	= h9E	
watch_dog_on	= b10011111	= h9F	
EEPROM Access:			
EEprom_bgap_off	= b10000110	= h86	
EEprom_bgap_on	= b10000111	= h87	
EEprom_enable_off	= b10010000	= h90	
EEprom_enable_on	= b10010001	= h91	
EEprom_read	= b10100000	= hA0	(读保护)
EEprom_write	= b10100001	= hA1	
EEprom_erase	= b10100010	= hA2	
EEprom_bwrite	= b10100011	= hA3	
EEprom_berase	= b10100100	= hA4	

在对 EEprom 进行读写之前要开启带隙基准准许进行操作。
发送 EEprom_bgap_on, EEprom_enable_on

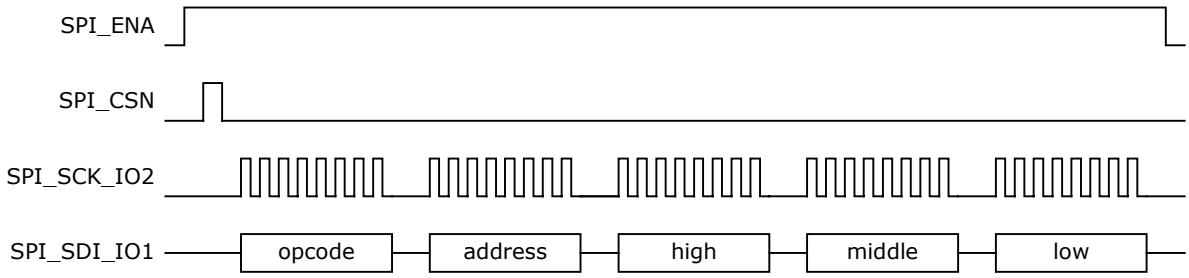
7.2.1 RAM 读时序

Figure 17



7.2.2 RAM 写时序

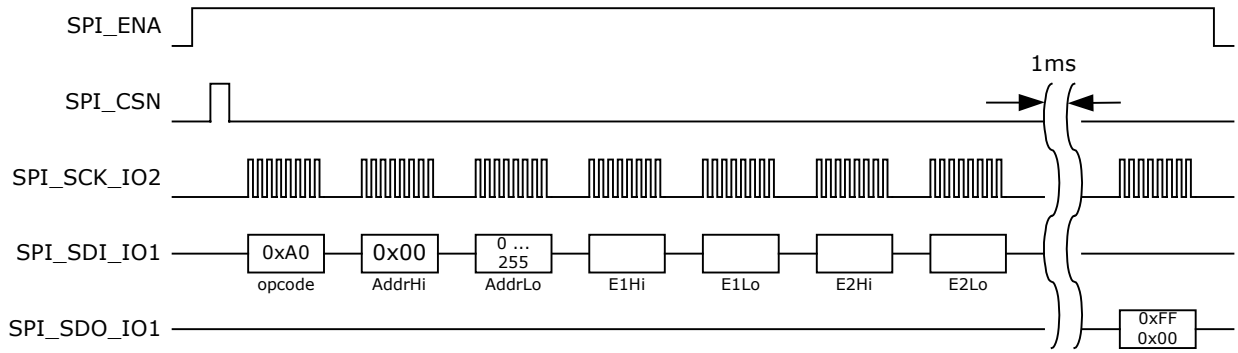
Figure 18



7.2.3 EEPROM 读进入 / 读保护

PSØ8 EEPROM 受到保护的，当在没有授权情况下内容是不能被读出的。仅仅当已知的内容与 EEPROM 中的内容完全相同的时候，才能够进行操作。读地址 0 到 255 的同时将会检查两个 EEPROM 的 16 位字. 因此命令 EEProm_read 是跟在 EEPROM1 高字节, EEPROM1 低字节， EEPROM2 高字节, EEPROM2 字节之后的, 那么芯片将会用所传输的数据与 EEPROM 的内容向比较。结果将会在 1 ms 后在 SDO 端口有效. =xFF 代表正确, 0x00 代表错误数据.

图 19

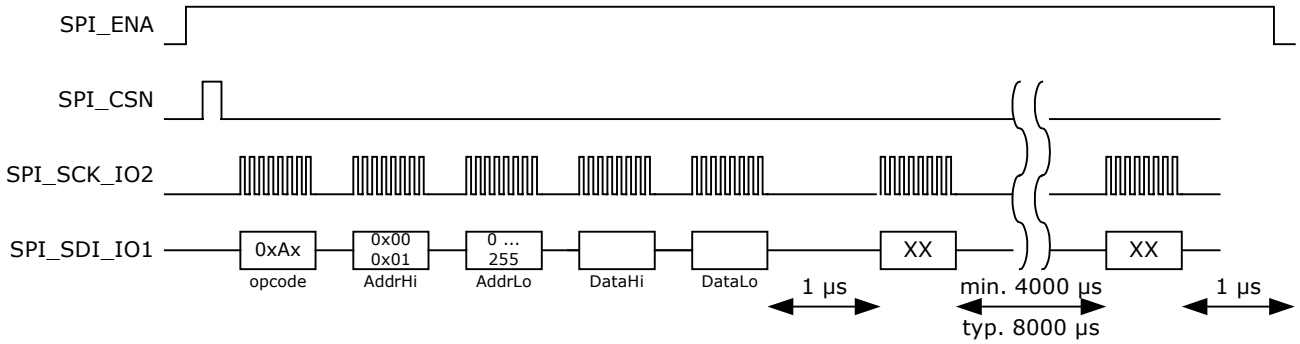


没有被授权的人如果想进行读操作必须要测试所有的组合。这样的尝试将会持续 $2^{31}*256*1ms$ = 17 年.

7.2.4 EEPROM 写进入

EEPROM 被分为 2 个 512 bytes 的模块，或者 2 个 256 字符的模块。那么操作不同模块是根据发送地址码的最低那个位而决定的。在发送写命令和地址码后跟随发送的是 16bit 数据的高 8 位，然后再发低 8 位。烧写 EEprom 过程是在第 6 个 8 位字写进之后开始的，那么至少在 4ms 后，再发送第 7 个 8 位字节写进之后烧写结束。那么第 6 个 8 位字节块和第 7 个 8 位字节块在这里所发送数值是无所谓的，只是为了控制 EEprom 的烧写。

图 20 EEProm_write



EEProm_erase: 这个命令是删除在不同模块（由 AddrHi 决定哪个模块）中 AddrLo 地址中的 16 位数据。与 EEProm_write 的过程差不多，但是所写的数据 DataHi DataLo 是不会真正写进地址，因为地址的内容要被删除。

EEProm_erase: 这个命令是删除整个模块的内容（由 AddrHi 决定哪个模块）。与 EEProm_write 的过程差不多，但是所写的地址内容与传输的数据内容都会被忽略，因为要删除整个模块的内容。因为 EEPROM 由两个模块组成，因此要删除整个两个模块要发送两次这个命令 (AddrHi = 0 & 1).

8 LCD-驱动

LCD 驱动有如下特性:

18 个管脚为

- 1/4 duty 最多 6 个数字包括小数点和 8 个特殊字符
- 1/3 duty 最多 5 个数字包括小数点和 5 个特殊字符
- 1/2 duty 最多 4 个数字包括小数点

将显示器的电压稳定在 3 V, 2.5V 和 2V

为 3 V 和 2.5 V 显示集成的电压倍压器

节能的 2 V 电源无需电压倍压器

可工作在非稳定电源下如锂电池或者太阳能电池

休眠状态下无电流消耗

驱动强度可以根据段码尺寸以及电流消耗进行调整

输出可调整因此可以连接已经连好的显示器

有 7 段码数字显示的转换表格

有 24 位数字转换的 ROM 代码

8.1 基本配置

带有 lcd_duty 的 LCD 被开启然后可以设置为特殊的 multiplex 模式

lcd_duty = 0 关闭
= 1 2x multiplex
= 2 3x multiplex
= 3 4x multiplex 模式

lcd_freq 控制了所有像素的开启时间。一个像素开启时间越长那么需要越少的电流。对于无闪烁的显示来讲推荐使用刷新率 > 30 Hz。因此开启时间则取决于所选择的 multiplex 模式。

lcd_freq[2:0] =	像素开启时间	Multiplex 模式 1/4	1/3	1/2
0	8.0 ms	15	20	31 Hz
1	4.8 ms	26	34	52 Hz
2	4.0 ms	31	42	62 Hz
3	3.2 ms	30	52	78 Hz
4	2.4 ms	52	69	104 Hz
5	2.0 ms	62	82	125 Hz
6	1.6 ms	78	104	176 Hz
7	1.2 ms	104	138	208 Hz

显示器有一个休眠模式。在这个模式里面显示器是被关闭的，但是产生电压被转换到高阻抗。因此可以非常快的将显示屏开启。这在自动开启模式中也许是一种不错的选择。

lcd_standby = 0 LDC 开启
= 1 休眠

8.2 LCD-电源

PS08 内部集成一个充电泵来加倍和稳定驱动 3 V 和 2.5 V LCD 显示。 2 V 的显示可以很容易的直接通过电源进行供电。外部电容的选择要根据显示器的尺寸或者其电容大小来决定。

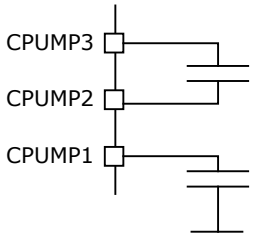
配置： **寄存器** 11,**位** 10，10：lcd_vlt

寄存器 16，**位** 19：lcd_direct_dirve

lcd_vlt[1:0] = 0 2.0 V
= 1 2.5 V
= 2 3.0 V
= 3 2.0V(**不带电压倍压**)

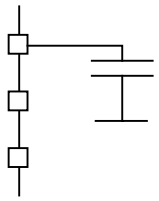
lcd_directdrive=1 lcd 直接被 Vcc 驱动，不带 charge pump 和电压调节。这将减少 LCD 电流，应用在太阳能模式下。

- 3 V 和 2.5 V 操作时带有电压加倍器



第一步两个电容都被充电到一半的显示电压， 1.5 V 或 1.25 V。 这个电压可以在管脚 CPUMP1 和 CPUMP3 被检测到。第 2 步两个电容被改变成串联接法。管脚 CPUMP3 将显示全部 LCD 电压，而管脚 CPUMP1 和 CPUMP2 显示了 LCD 的一半电压。

- 2 V 工作电压不带电压加倍器



在这个模式下 lcd 电压被从一个不稳定的电源电压通过电容的充电稳定到 LCD 电压。供电电压在这个模式下不能够降到 2V 以下。

- 直接驱动模式

3 个选择是在没有对电压调整的情况下直接通过电源进行供电。因此不需要外部电容连接，输出被设置为低阻抗输出。这是通过如下设置完成的：

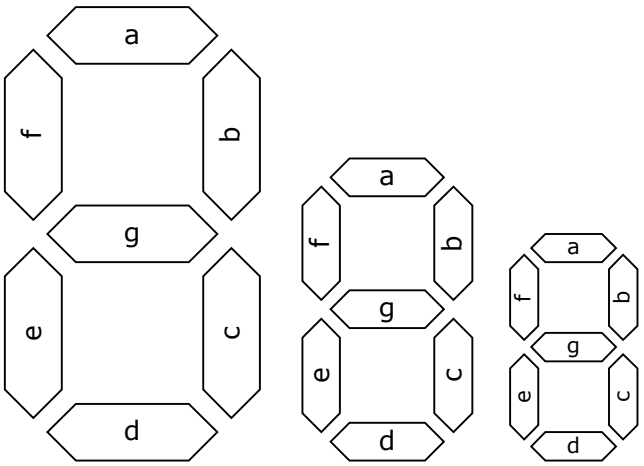
lcd_r_const = 0 (10 kOhm).
lcd_vlt[1:0] = 0 2.0 V

8.3 LCD 输出驱动配置

输出驱动的内部电阻可以根据显示器的尺寸进行调整。通过这个方法可以进一步将电流消耗优化。显示器的尺寸影响了：

电容器的尺寸
驱动的内部电阻值
充电泵的内部最小充电时间
显示器重新开启的时间

图 21



配置寄存器

控制位	大	中	小	功能
lcd_fastld[1:0]	3	2	1	配置了快速充电的周期数(10ms) 带有低欧姆分压器
lcd_swload1k	1	1	0	0 = 通过200欧姆电阻给电容充电 1 = 通过1K欧姆电阻给电容充电
lcd_r_const[1:0]	1	2	3	设置了LCD的跨越电阻值驱动 0 = 15 k 1 = 200 k

				2 = 800 k 3 = 1600 k
lcd_charge[1:0]	0	2	3	在重新充电前选择需要等待多少个LCD时钟周期 0 = 每个周期 1 =两个周期 2 = 4个周期
lcd_r_fastld	1	2	3	配置快速充电周期 (10ms) 带有低欧姆分压器

LCD 驱动方法
在每个模式输出驱动都有4个水平， 0， 1/3， 2/3和完全LCD电压。

图 22

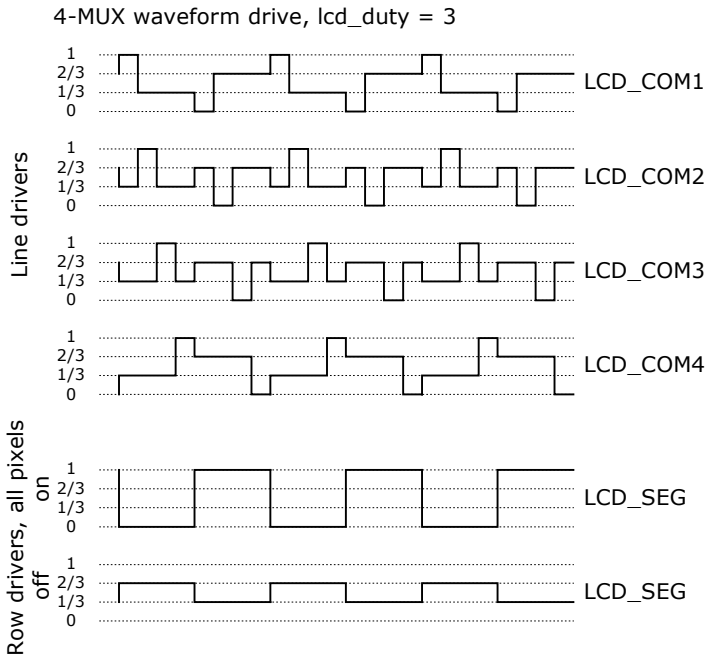


图 23

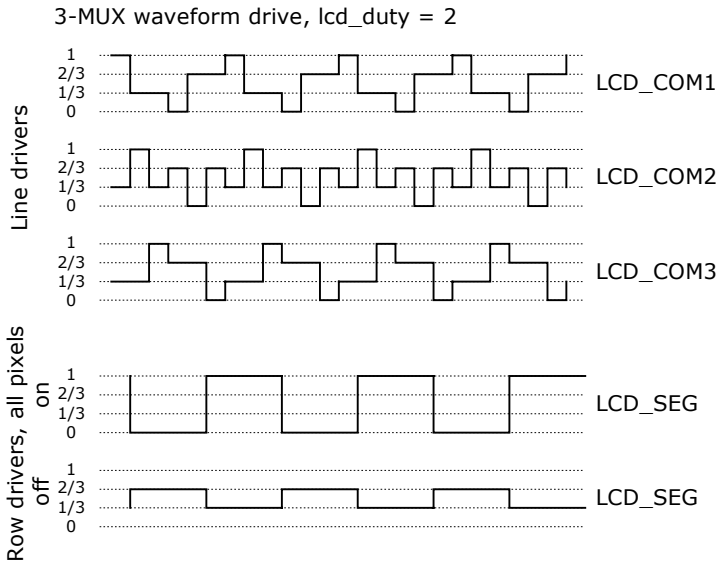
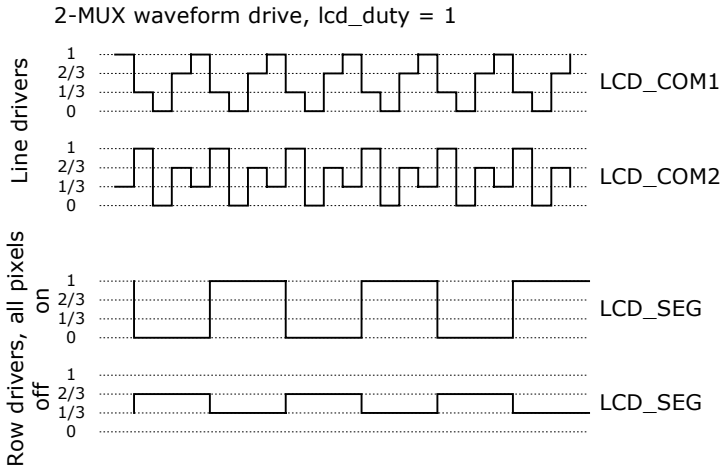
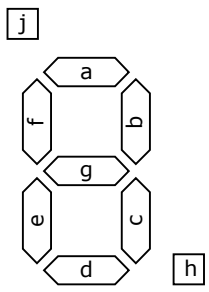


图 24



8.4 LCD 控制

8 个部分组成了一个数字。每个部分根据 a 到 g 的字母被命名。小数点被命名为 h。



每个部分被开启或者关闭通过设置寄存器 13, 14 和 15 中的相应位到 1 或者 0。这个分配与 multiplex 模式无关。如下表所示:

Table 7

数字	段码		Hex 值
	hgfe	dcba	
"0"	0011	1111	3F
"1"	0000	0110	06
"2"	0101	1011	5B
"3"	0100	1111	4F
"4"	0110	0110	66
"5"	0110	1101	6D
"6"	0111	1101	7D
"7"	0000	0111	07
"8"	0111	1111	7F
"9"	0110	1111	6F

在配置内存中的位置
在 2x multiplex 模式下 lcd_segment 的低 32 位被应用。
在 3x multiplex 模式下每个数字被表示为一个 3x3 的矩阵，包括一个额外的特殊字符。lcd_segment 的低 40 位被用作显示 5 个数字。特殊符号需要对 40-44 位进行控制。

表 8

数字	lcd_segment	控制寄存器	应用
6	[55:48]	15	1/4
5	[47:40]	14	1/4, 1/3
4	[39:32]	14	1/4, 1/3
3	[31:24]	14	1/4, 1/3,1/2
2	[23:16]	13	1/4, 1/3,1/2
1	[15:8]	13	1/4, 1/3,1/2
0	[7:0]	13	1/4, 1/3,1/2

在 dez2lcd (D)里面 有一个处理器的将十进制数据转换成 0-9 字符的特殊代码。它将地址累加器的最低 4 位代码转化成标准 7 段码。

为了进一步更方便， 在 ROM 代码中有一个完全的 24 位数字转换的子程序。
在汇编程序中这个子程序是以操作码 no2lcd 出现的。在 X-累加器的数值被转换然后被写入 LCD 内存的低 48 位(lcd_segment[39:0]中去。带有符号的原始数值被写回到 X-累加器然后可以用来设置在显示器上的符号。小数点的位置在 Y-累加器中被显示。小数点前面的 0 不会显示。LCD 驱动忽略了上面在 2xmultplex 中的两个数字和在 3xmultplex 中的上面的数字。在 3x 和 4xmultplex 中的特殊字符将不能被改变 (lcd_segment[55:48])。 在 2xmultiplex 中显示的小数点可也用于特殊字符。在这个情况下他们必须在转换之后重新存储。

注：必须要分别通知 LCD 驱动关于在 lcd 寄存器 13-15 中的新数据。这个是通过操作码 newlcd 来完成。

样例程序:

```
move   x , r        ; 加载结果到 x-累加器
move   y , 2              ; 加载小数点的位置到 y-累加器
jsub   no2lcd       ; 转换到 7-段码显示格式
       newlcd       ; 刷新 LCD
       clrwdt       ; 重置看门狗定时
stop                ; 停止 uC
```

8.5 连接原理图

图 25 4-MUX (¼ duty)

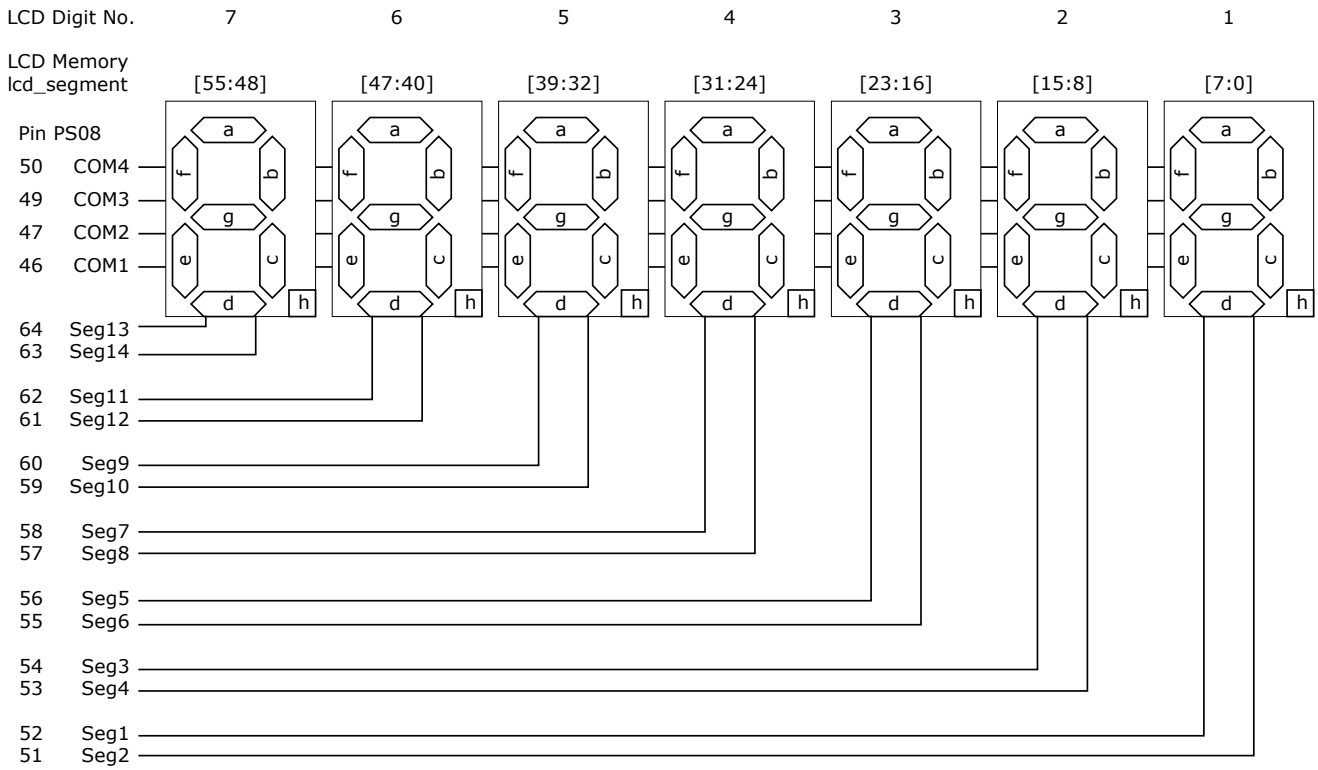


图 26 3-MUX (1/3 duty)

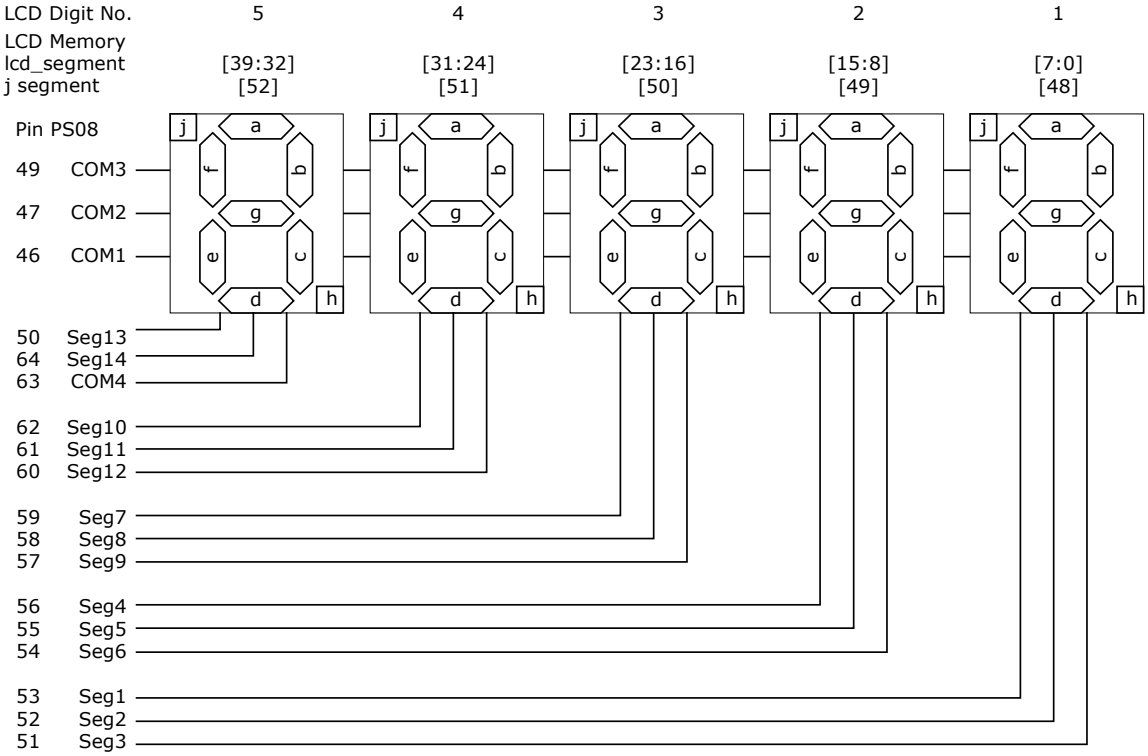
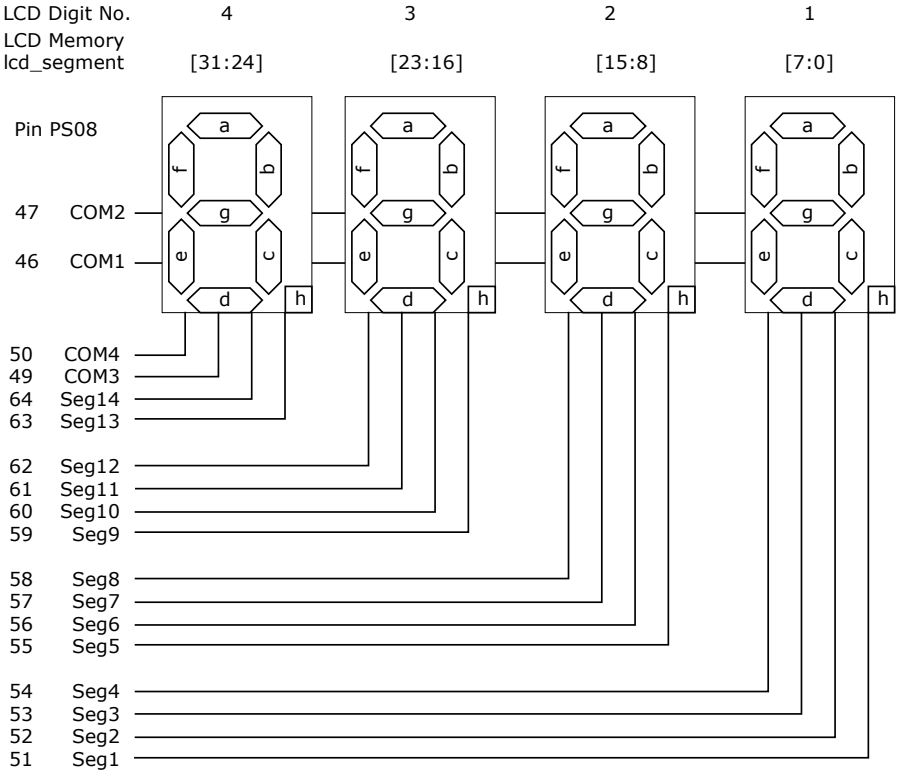


图 27 2-MUX (½ duty)



8.6 设置各段位置

每一个 lcd_segment 控制位的段都可以任意连接到公共驱动线和段码驱动线上。这一点提供了高的驱动灵活性而且允许连接现有的不同的 LCD。

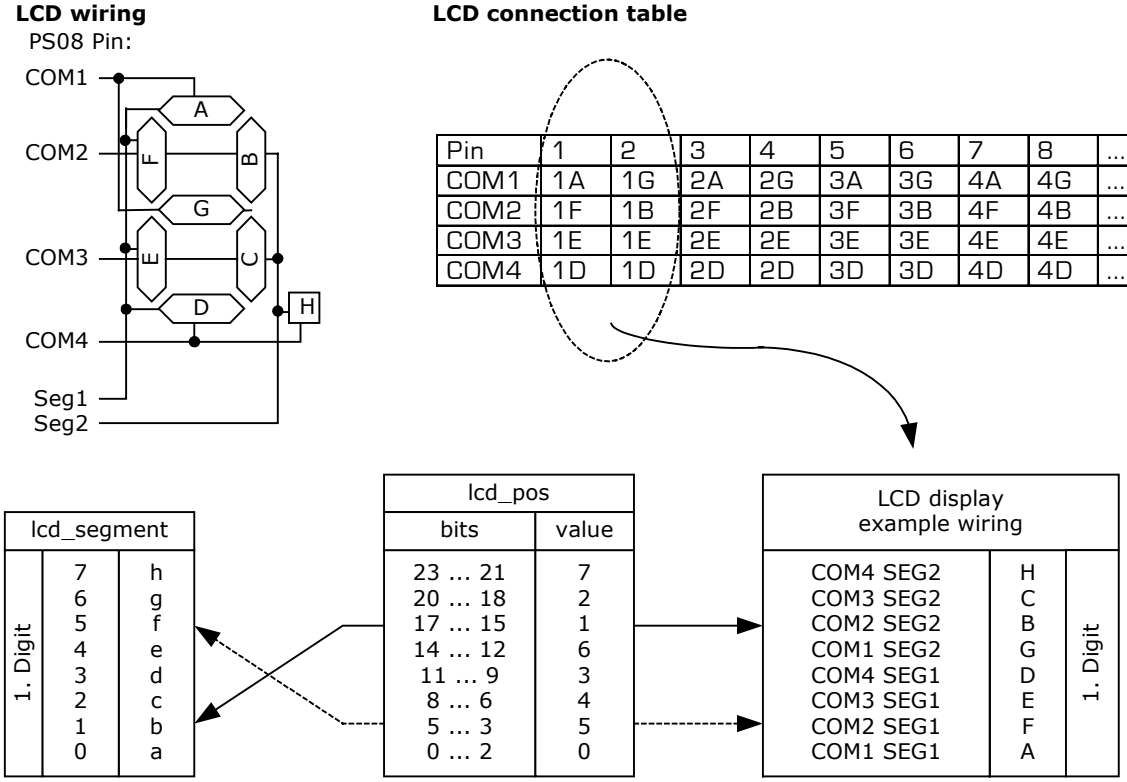
限制:
段码线和实际的公共线 3,4 必须要连接在正确的数字上。 在一个数字里面的顺序可以是随意的。否则，no2lcd 将会搞混所有数字。

在寄存器 lcd_segment 中程序设置了被显示的各段。在 lcd_pos 中则定义了在 lcd_segment 中具体哪一位对应 8 个目标段的位置。

表 9 lcd_pos

lcd_segment[] bits	所显示段	lcd_pos[] (选择 Seg&Com 交叉点)
7, 15, 23, 31, 39, 47, 55	h	23 to 21
6, 14, 22, 30, 38, 46, 54	g	20 to 18
5, 13, 21, 29, 37, 45, 53	f	17 to 15
4, 12, 20, 28, 36, 44, 52	e	14 to 12
3, 11, 19, 27, 35, 43, 51	d	11 to 9
2, 10, 18, 26, 34, 42, 50	c	8 to 6
1, 9, 17, 25, 33, 41, 49	b	5 to 3
0, 8, 16, 24, 32, 40, 48	a	2 to 0

例如下图:



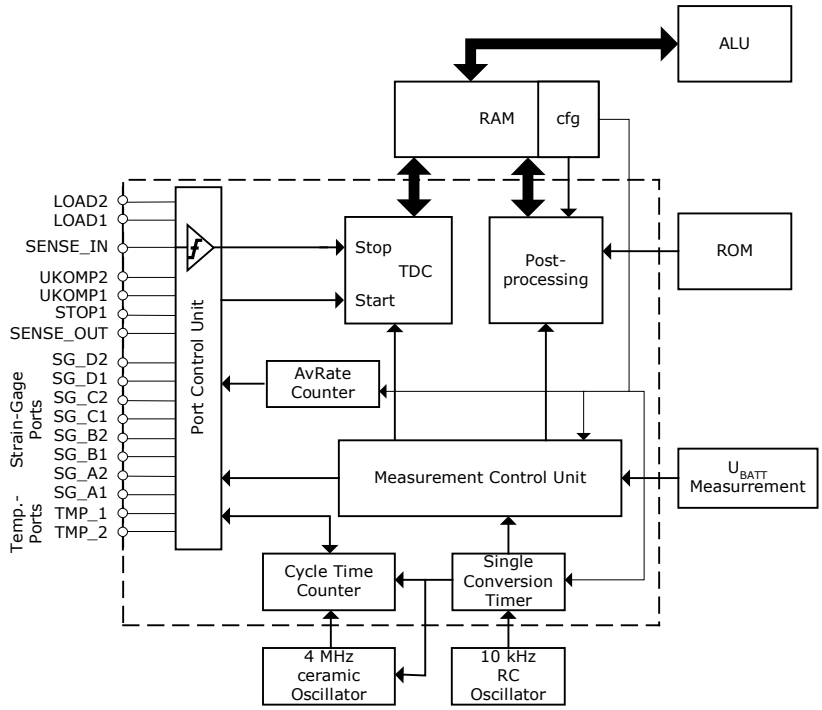
PSØ8 通过简单的尝试来调整 LCD 段码。

方法如下:

1. 开启 LCD
2. 设置 lcd_segment[] 位 0 = '1'在第一个数字中显示段 'a' (config_reg 地址 61)
3. 设置所有 lcd_pos 位到 1
3. 将 0 从 lcd_pos[2:0] 移动到 lcd_pos[5:3] 再移动到 lcd_pos[8:6] ... 直到正确的段被显示出来.
4. 选择 lcd_segment[] 位 1 = '1' 在第一个数字中显示段 'b' (config_reg 地址 61)
5. 将 1 从 lcd_pos[2:0] 移动到 lcd_pos[5:3] 移动到 lcd_pos[8:6] ... 直到正确的段被显示出来. 但是要保持之前你已经设置好的段 a 的 lcd_pos 位。
... 如此测试知道所有位都被正确显示在正确的位置上

9. 高尖端转换器

图 28



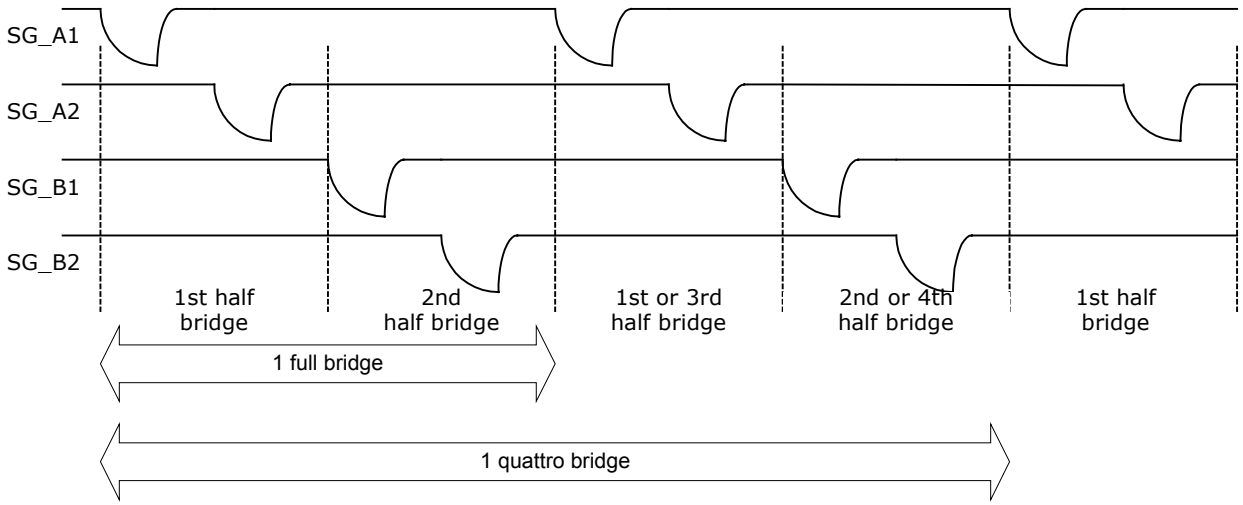
9.1 测量原理

PICOSTRAIN 基础转换器有 4 个独立的测量半桥的端口。两个半桥组成一个全桥或者一个经典的惠斯通电桥。

作为一个附加的功能芯片还提供了温度测量端口，通过连接 KTY 或炭阻温度电阻可以进行温度测量。比如您所获得的应变传感器并没有温度补偿措施，那么可以通过 RSPAN_BY_TEMP 和温度测量来对传感器本身进行温度补偿。

应变本身是通过放电时间的测量而得出的。放电时间是由应变电阻和充电电容 Cload 共同定义的。

图 29



推荐的 Cload 的值为:

Rsg = 350 欧姆: Cload ~ 300 nF 到 400 nF
Rsg = 1000 欧姆: Cload ~ 100 nF 到 150 nF（33nF 在需要超低功耗情况下）

推荐的电容种类：
x7R 对于低端的应用
COG 或 CFCAP 系列 对于高端衡器应用
无需很窄的容忍度。传感器变化大于 20%也是没有问题的。进一步的信息请参见 9.5 章节。

测量单元控制了高速 4M 晶振时钟的启振，而且可以测量操作电压。

最后它将进行数据处理然后将最终结果输入到 RAM 里面。

9.2 模式和时序

一共有 4 个操作模式，在放电时间的时序上有所不同。

- 持续工作模式 这个是标准模式, 所有 > 500 µA 电流消耗的应用
- Stretched 持续模式 1 & 2
- 单一转换模式 超低功耗，低采样频率
- Stretched 单一转换模式 1&2 非常低功耗, 过采样

图 30

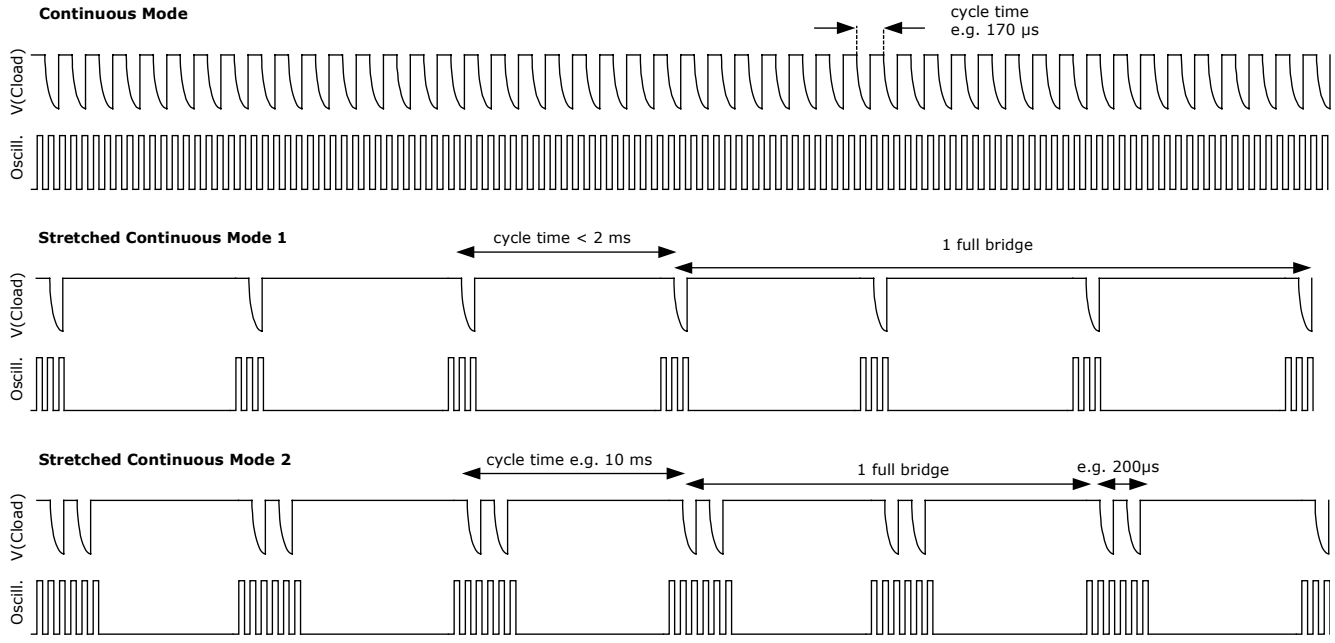
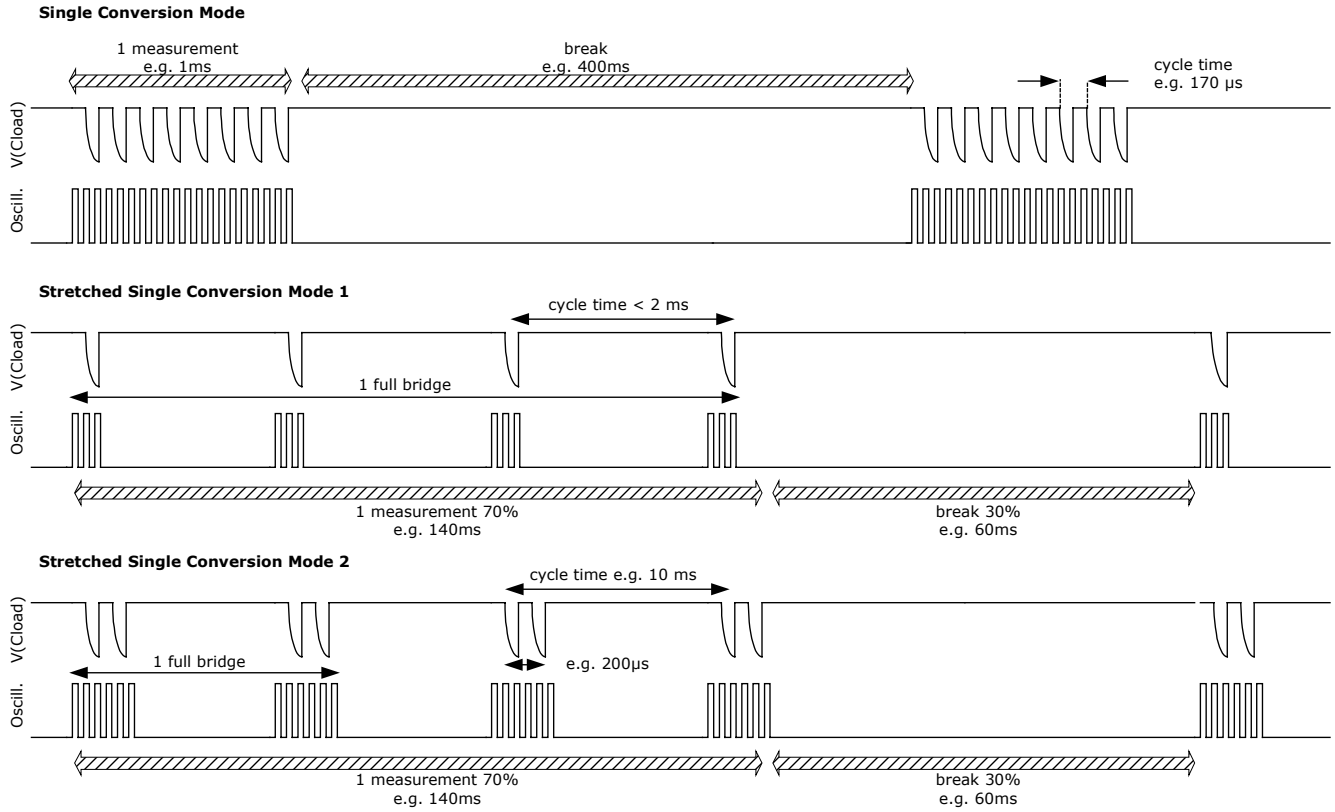


图 31



4 个主要参数定义了不同的模式:

- 单一转换　　在持续操作模式和单一分步测量模式中选择.
- stretch　　选择 4Mhz 晶振在测量中实时开启或者只在放电循环时候才开启
- cycletime　　定义了一个或者一对放电时间过程
  以 4 MHz 时钟为基础的或者在 stretched 模式下以 10 kHz 时钟为基础
- avrate　　定义了构成一次完整单词测量的比例测量次数 [内部平均].

9.2.1 single_conversion（单一转换）

配置:　　寄存器 2, 位 2: single_conversion

single_conversion = 0　　选择持续测量模式.在这个模式里面 PSØ8 是持续进行测量的。
4M 晶振是持续工作的。. 这样的电流消耗大概为 130 µA @ 3.0 V.

single_conversion = 1　　选择了单一转换模式.在这个模式下 PSØ8 将会在一次完整测量后关闭 4M 晶振，关闭晶振的持续时间由单一转换的计数器来决定。

9.2.2 stretch（拉伸模式）

配置:　　寄存器 3, 位 12, 13: stretch

stretch = 0　　关闭

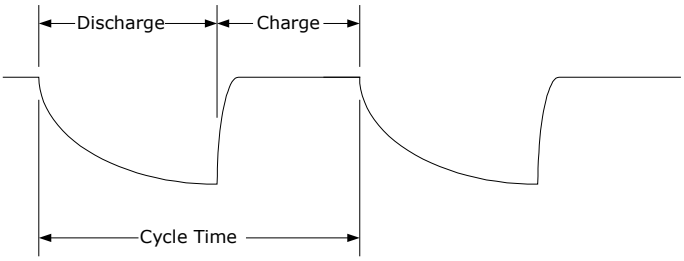
stretch = 1　　4M 晶振仅在放电的测量时开启。Cycletime（在两个放电时间之间的时间）是通过 10 kHz 来计算的

stretch = 2 or 3	4 MHz 晶振仅在一个半桥测量的放电持续时间内开启 (两次放电时间测量). cycle time (板桥测量之间的时间) 是以 10 kHz 晶振为基准的。在两个放电时间测量之间的时间间隔为 200 µs （stretch = 2 ）或者 300 µs（stretch = 3）

9.2.3 cytime (Cycle Time)

Clycletime 是 PS08 内部用于给 Cload 充电和放电的时间和。下面的图说明了这个关系。
纵坐标代表 Cload 端的电压。

图 32 在 Cload 端的电压



放电的时间是由金属应变电阻和选择的电容 Cload 所决定的。如果应用我们的推荐值 (例如 400 nF 和 350 Ohm 金属应变电阻)那么放电时间会在 80 - 120 µs 范围内。充电时间必须要足够长以保证对于 Cload 的充电完全。最小充电时间我们推荐为整个充放电时间总和的 30% (放电+充电时间)。那么 cycletime 的上限是有设置 cycletime 的寄存器的极限来限制的。
配置： 寄存器 2，位 4 到 13：cytime

cycletime 由外部高速时钟频率除以 8 决定，或者在 stretch 模式下是通过 10KHz 来决定的。

例如:
持续测量模式: CYTIME=80 →80*8*250ns=160us 在 4MHz 的情况下
CYTIME = Cycle time[us]/2us
Stretched 持续测量模式: CYTIME = 10 → 10 *100 us = 1 ms

如果应用推荐的 Cload 值那么 cycletime 不能降到 150µs 以下。

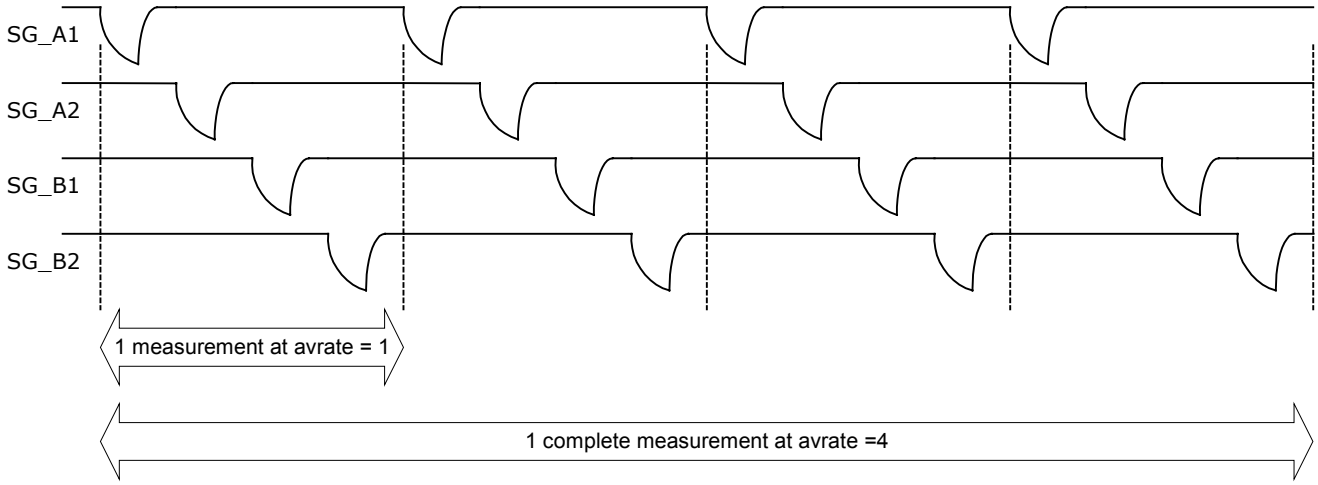
9.2.4 avrate (平均率)

通过设置内部的平均率，PS08 的精度将会被提高。
配置 寄存器 2，位 14-23：avrate

平均率设置了在一次完整的测量循环内需要测量多少次应变电阻和参考电阻的比率。一次 AVRate 的 cycletime 的数目根据所选择的半桥数而定。对于每个半桥需要两次 cycletime。

全桥 →每个 AVRate 4 次 Cycle Time
4 角桥 →每个 AVRate 8 次 Cycle Time

图 33 例: Full 全桥



9.2.5 模式选择标准

应用	模式		参数	描述
高精度不考虑功耗问题 标准模式适用所有工作电流 > 500 µA	持续模式	持续测量模式	single_conversion = 0 stretch = 0 cycle time = cytime*8*250 ns	持续测量模式, 4 MHz 晶振持续工作
高精度低功耗		Stretched 持续模式 1	single_conversion = 0 stretch = 1 cycle time = cytime*100 µs	持续工作测量模式. 4 MHz 晶振仅在放电时间被开启。cycle time 要设置为小于 2 ms 来避免欠采样现象。
高精度低功耗		Stretched 持续模式 2	single_conversion = 0 stretch = 2 or 3 cycle time = cytime*100µs	持续测量模式. 4 MHz 晶振仅在放电时间被开启。 在采样频率> 100 Hz 时它的功耗大于模式 1.
低功耗应用 机械振动相对稳定的应用如压力传感器	单一转换模式	单一转换模式	single_conversion = 1 stretch = 0 cycle time = cytime*8*250ns	可选择的配置最低功耗, 欠采样 -> 没有对机械振动的一致及优化处理
高精度而且低功耗的应用, 例如电池驱动的 OIML3000 的衡器		Stretched 单一转换模式 1	single_conversion = 1 stretch = 1 cycle time = cytime*100µs	可选择非常低功耗测量以及过采样来抑制机械振动。 cycle time 应该设置小于 2 ms 来提高采样频率.当在 70%放电, 30% 充电时间情况下性能最优
高精度而且低功耗应用 例如太阳能电池板驱动衡器		Stretched single conversion mode 2	single_conversion = 1 stretch = 2 or 3 cycle time = cytime*100µs	可选择的低功耗测量及过采样来抑制机械振动。 在基本采样频率 > 100 Hz 的时候它的功耗比模式 1 要大

9.3 性能设置

9.3.1 精度 和 AVRate

所选择的一次测量的 AVRate 同时决定了测量的精度。AVRate 越高那么测量的精度相对也就越高。测量精度和 AVRate 这件的关系为测量精度会提高 AVRate 的均方根倍。也就是说精度将会提高整个 cycletime 数的均方根倍。

对于精度的计算:

在 AVRATE =1 时一个半桥和在快速模式下推荐的放电时间以及 2mV/V 灵敏度时的基础精度为 :

带有内部比较器: 13.3 Bit eff.
带有外部双极性比较器: 13.8 Bit eff.

在设置高的 AVRATE 值时精度的计算如下:

$$\text{Re}\,solution = \text{Re}\,solution[AVRate = 1] + \frac{\ln\left(\sqrt{AVRate * Brige - factor}\right)}{\ln(2)}$$

Bridge-factor 为:
2 为全桥情况
4 为 4 驱桥情况

例 1:

AVRate = 12, 四区驱桥, 内部比较器

精度 = 13.3 + LN(√(12*4))/ln(2) = 13.3 + 2,8 = 16.1 有效位. = 70.000 有效分度 = 10,000 峰峰值分度 在快速模式下。

例 2:

AVRate = 450, 全桥, 外部比较器

精度 = 13.9 + LN(√(450*2))/ln(2) = 13.8 + 4.9 = 18.7 Bit eff. = 425,000 有效分度= 70,000 峰峰值分度 在快速模式下。

9.3.2 转换时间/测量刷新率（持续测量模式）

一次完整测量的时间可以通过下面的公式进行计算：

Tconversion = CycleTime*(2*AVRATE * Bridge-factor + 6 +MFake*2 + 1)

Mfake = #伪测量数目 温度测量开启

Mfake-寄存器	#伪测量数目
0	0
1	2
2	4
3	16

针对上面的计算的例子:

例 1:
Cycle time = 110µs
AVRate=12
四驱桥
Mfake=1
Tconversion = 110µs*(2*12*4+6+2+1) =11.55 ms → 最大测量刷新率为 86.6 Hz

例 2:

Cycle time = 110µs
AVRate=450
全桥
Mfake=2

Tconversion = 110µs*(2*450*2+6+4+1) =199.21 ms →最大测量刷新率为 5,02 Hz

9.3.3 转换时间 / 测量频率 (单一转换模式)

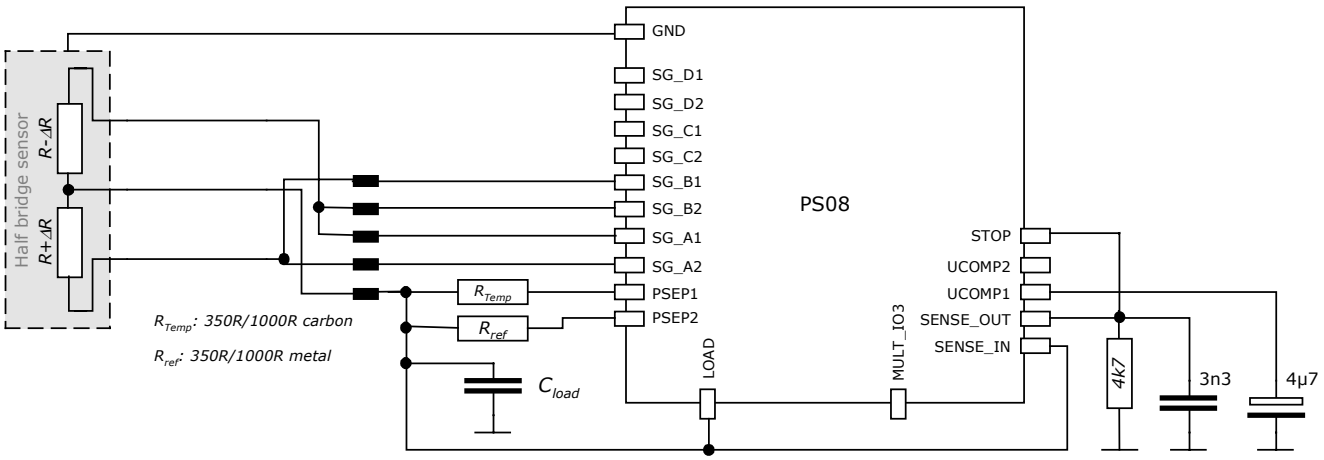
如果 PS08 被配置为在 单一转换模式 (在 configreg_02 中的位 4),测量频率是通过在 configreg_00 中的 tdc_conv_cnt[23:16] 设置的。这个值直接关系到转换的时间 (乘以 6.4ms).

例如:
configreg_00: 0x158200 → tdc_conv_cnt[23:16] = 0x15 = 21 十进制
→ 21 x 6.4ms = 0.1344 ms
→ 测量频率 = 1 / 0.1344ms = 7.44 Hz
请注意,如果您应用单一转换模式，需要一次完整测量的时间必须要与所设置的转换计数器的设置时间相适合 (tdc_conv_cnt).

9.4 连接应变传感器

9.4.1 半桥模式（连接如全桥）

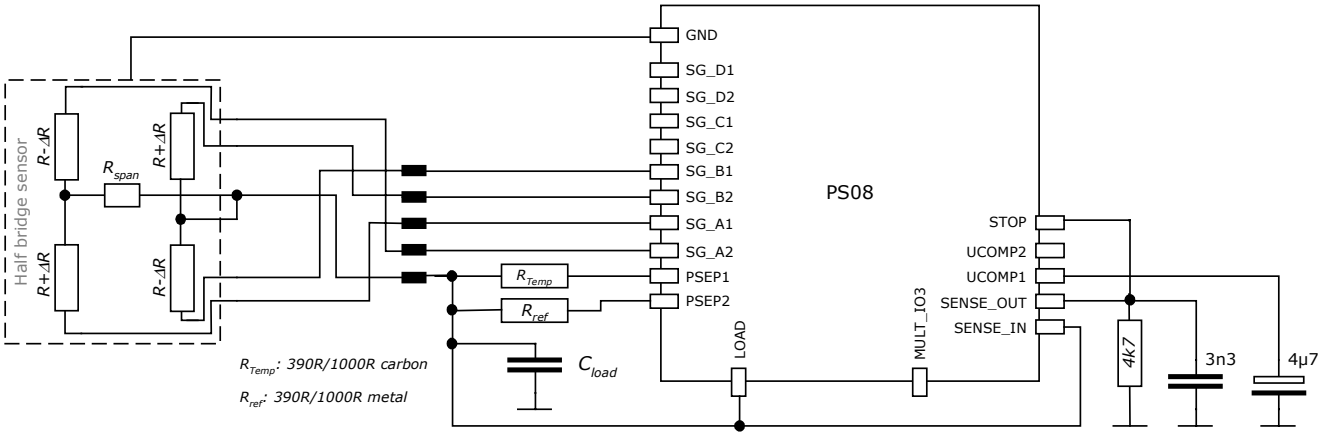
图 34



注:
为了得到更好的温度漂移特性，我们推荐同时将半桥连接到端口 B。 这个连接相应为全桥连接需要在寄存器 03 中进行相应设置 bridge[1:0]=1 (2 个半桥)。
乘法因数应该有设置相反的符号。比如 Mult_Hb1 = +1, Mult_Hb2 = -1。

9.4.2 全桥模式

图 35

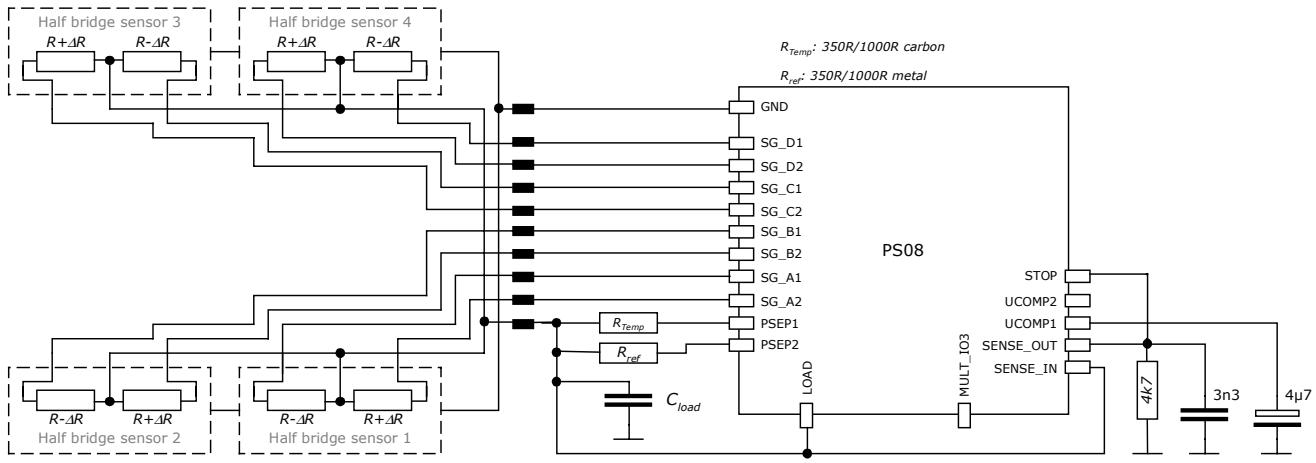


Note:
注:
为了得到更好的温度漂移特性，我们推荐同时将半桥连接到端口 B。 这个连接相应为全桥连接需要在寄存器 03 中进行相应设置 bridge[1:0]=1 (2 个半桥)。
乘法因数应该有设置相反的符号。比如 Mult_Hb1 = +1, Mult_Hb2 = -1。.

9.4.3 四桥模式

(四个传感器)

图 36



注：

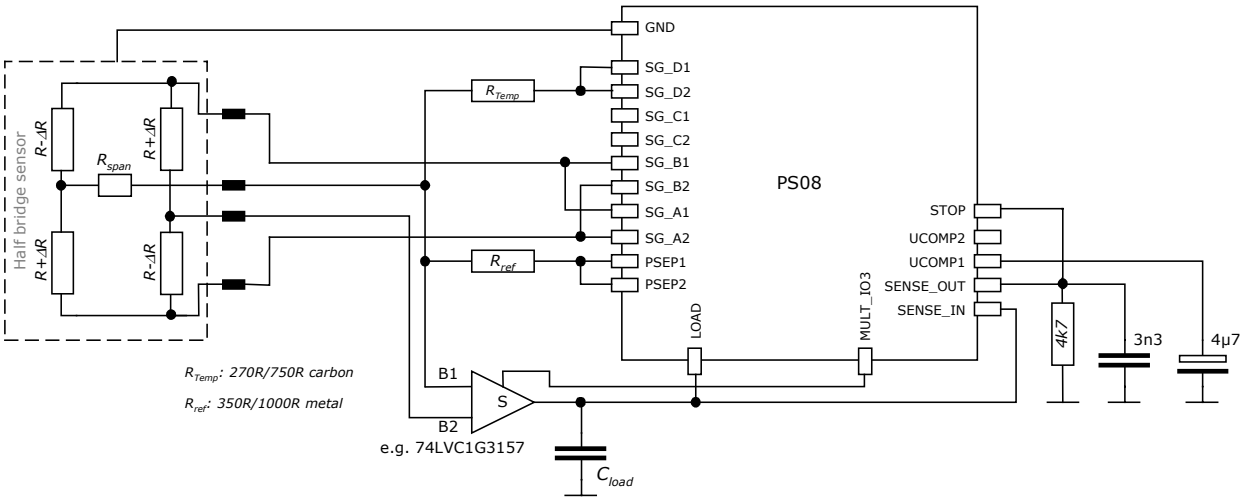
在这个模式下所有的乘法系数要有相同的符号。

在寄存器 2 种设置 bridge[1:0] = 3。

9.4.4 惠斯通电桥模式

当应用惠斯通电桥模式需要一个如 74LVC1G3157 的模拟开关。

37

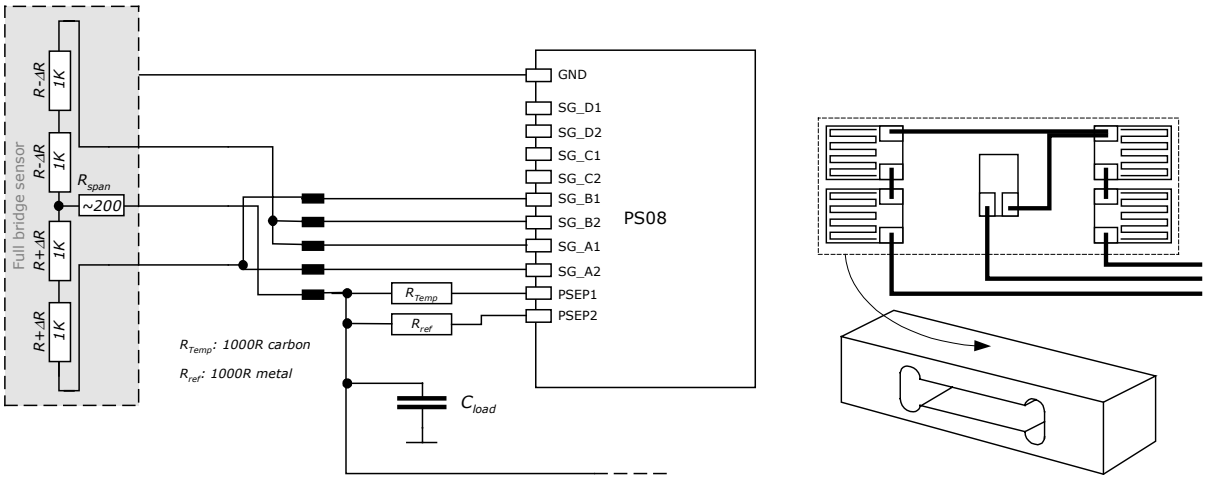


注:

在惠斯通桥接模式下系统会有 0.6 位的精度损失。因此我们仅推荐在初次测试惠斯通桥接传感器时和长线应用(> 1 m)的时候使用惠斯通桥接模式。

配置通过设置寄存器 2 的 bridge[1:0] = 1。

9.4.5 全桥连接成半桥低功耗



注:

这样的连线是非常适合太阳能电池板驱动的应用的. 半桥的电阻被加倍到 2K 欧姆。通过传感器的电流消耗将会减少一半。

为了得到更好的温度漂移特性，我们推荐同时将半桥连接到端口 B。 这个连接相应为全桥连接需要在寄存器 03 中进行相应设置 bridge[1:0]=1 (2 个半桥)。

乘法因数应该有设置相反的符号。比如 Mult_Hb1 = +1, Mult_Hb2 = -1。

9.5 充电电容 (Cload)

这个充放电电容是电路中一个非常重要的部分直接影响到测量的质量和温度稳定性。因此，我们推荐下面的数值和材料：

Rsg = 350 Ohm → Cload ≈ 470 nF – 510 nF
Rsg = 1000 Ohm → Cload ≈ 150 nF – 180 nF

推荐的材料:

- COG* [高精度测量]
- CFCAP®太阳诱电多层陶瓷电容
- X7R [在温度稳定性上有一些损失]
- Polyester [在温度稳定性上有一些损失]

我们**不推荐**使用 ZOG 材料的电容！

*COG 电容容值最高可达 100nF 的电容器可在 Murata GRM31 系列中找到

注:

COG 电容在高端应用中是绝对的第一选择 [比如 6k 分度 OIML 或者更高标准的衡器]

CFCAP 对于高端衡器和ＯＩＭＬ衡器来说也是不错的选择

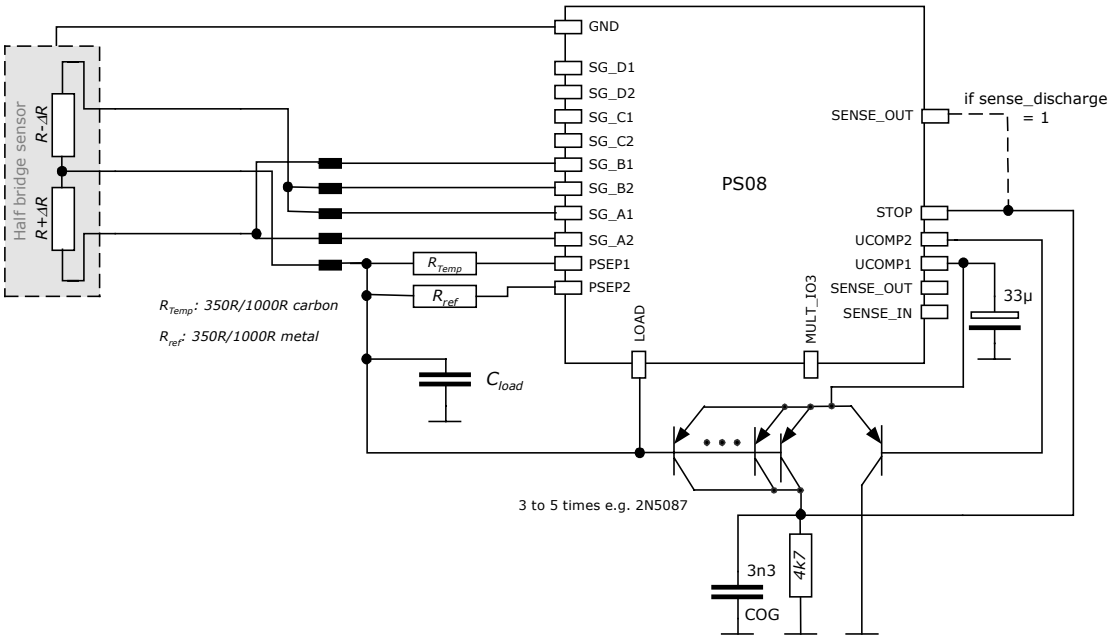
X7R 由于价格比较低因此是消费秤的第一选择。但是这个材料的电容会带来额外的温度 Gain 漂移[< +10C]

对于普通精度的消费衡器来说还有很多其他材料的电容也很适合 [比如 Polyester].

9.6 比较器

放电循环的结束是由一个比较器控制的。PS08 提供了内部的比较器通过设置寄存器 11 sel_compint 为 1。应用内部比较器在 2 mV/V 传感器，5Hz 刷新率时以及开启 SINC3 滚动平均时最高可以达到 60,000 峰峰值分度。
通过外部的双极性比较器可以更进一步的提高测量的精度。在同样情况下可以提高 30%的精度到 1 2 万分度。
加外部比较器电路图:

图 38



推荐:

- 低噪声 PNP 三极管如应该使用 2N5087 或 BC859
- 在 LOAD 端应该使用 5 个 并联的三极管
- 无需对三极管进行匹配
- 应用 COG-类型电容来做低通滤波电容

什么时候应用外部比较器?

有三个原因:

a) 需要测量很高精度

当客户在其应用中需要最佳精度的时候，例如计数称应用中。

b) 极低功耗情况

当用户在其应用中需要非常低的电流情况时，例如当需要做太阳能驱动衡器的时候。由于低噪声，AVRate 可以被减小到给定的精度，因此操作电流就被降低了，应用双极性三极管操作电流可以被几乎减小一半。

c) 超低电压

当客户希望他的应用中的电压低于 2.1 V Vdd, 比如说应用 1.55 V 小的纽扣电池的时候。 这里双极性三极管会带来非常好的效果 (使精度增加 0.5 位)。

9.6.1 比较器控制

比较器出于省电的原因可以仅在测量的时候被开启或者也可以持续开启(con_comp[1:0])。另外，内部比较器的工作电阻可以被改变通过寄存器设置(sel_compr[1:0]).

我们推荐下面的设置:

CON_COMP = 'b10 → 除了 load 外在测量时开启

SEL_COMPR = 'b00 → 7k 电阻被选择

如果 CON_COMP 被设置为 'b11 (持续开启) 比较器需要大约. 130µA@ 3.0 V 持续电流。

在 UCOMP1 和 STOP 端的电容器

在 UCOMP1 和 STOP 端的电容器对于低功耗来说是非常重要的。为了达到最佳效果我们推荐 33µF 给 Cucomp1 和 2.2 - 2.7nF 给 Cstop。请应用 COG 类型的电容给 C_{STOP}。对于 Cucomp1 可以用一个比较普通的电解电容。

如果应用内部比较器的话 C_{UCOMP1} 和 C_{STOP} 必须要被连接，而且电阻要 4.7k 欧姆的。

尽管如此，小一些的电容值也是可以的。

推荐数值:

Cucomp2:	不要小于 1µF
Cstop2:	低于 Cucomp2/3000

例如:
Ucomp2 = 1µF → Cstop2 < 1µF/3000 → 选择 330 pF。
噪声将会稍有增加，大概增加 0.2 – 0.3 位。

9.7 Rtemp / Rref

需要两个电阻 Rtemp 和 Rref，有两个原因
- 对于比较器延迟时间的纠正
- 温度测量

这两个电阻是必须要连接的。在没有温度测量的情况下两个电阻可以为同一种类型(比如炭阻)。

9.7.1 比较器延迟的纠正

因为比较器主要是考虑到低功耗应用，那么比较器的延迟时间是不能够被忽略的。这个延迟时间是和温度有关系的会造成一个增益误差，这个误差对于衡器应用的精度是有非常大的影响的。Rtemp 和 Rref 在操作中被应用于周期性的测量这个延迟时间。PS08 通过测量这个延迟时间来纠正测量结果中由这个延迟所引入的误差。

比较器的延迟时间与 Cucomp2 和 Cstop2 有关系。因为这些值可以被用户所改变。可以通过寄存器 Mult_PP[7:0]来调整校正的路线。

对于推荐的 Cucomp2 和 Cstop2(33µF 和 2.7 nF) 比较理想的值为 0x1A0 (十进制 160)。 如果电容器的值增加那么校正 Mult_PP 也必须增加，反之亦然。 如果选择的 Mult_PP 值太低那么增益误差将会随温度的升高或者电压的下降而降低。如果选择的 Mult_PP 值太高那么增益误差将会随温度升高或者电压降低而增加。

在选择了正确的 Mult_PP 值之后电路的增益将会在一个非常宽的电压和温度范围内稳定。 增益误差的温度漂移小于<1ppm/K。电源抑制比(PSRR) 大于 130 dB.

9.7.2 温度测量

温度测量通过测量两个电阻的放电时间的比率来完成的。其中一个电阻为温度敏感电阻，另一个为温度稳定电阻。敏感电阻可以是 KTY 类型或者更便宜的碳膜电阻。参考电阻可以为金属膜电阻。

9.7.3 Rtemp 和 Rref 的值

Rtemp 和 Rref 的值必须要调整到应变电阻的阻值和相应的桥接方式。

因此电阻应该有如下阻值:

一般 (半桥，全桥，四驱桥):	R = Rsg	(例如 1000 Ohm 和 1000 Ohm 桥)
惠斯通电桥:	R = 0.75*Rsg	(例如 750 Ohm 和 1000 Ohm 桥)

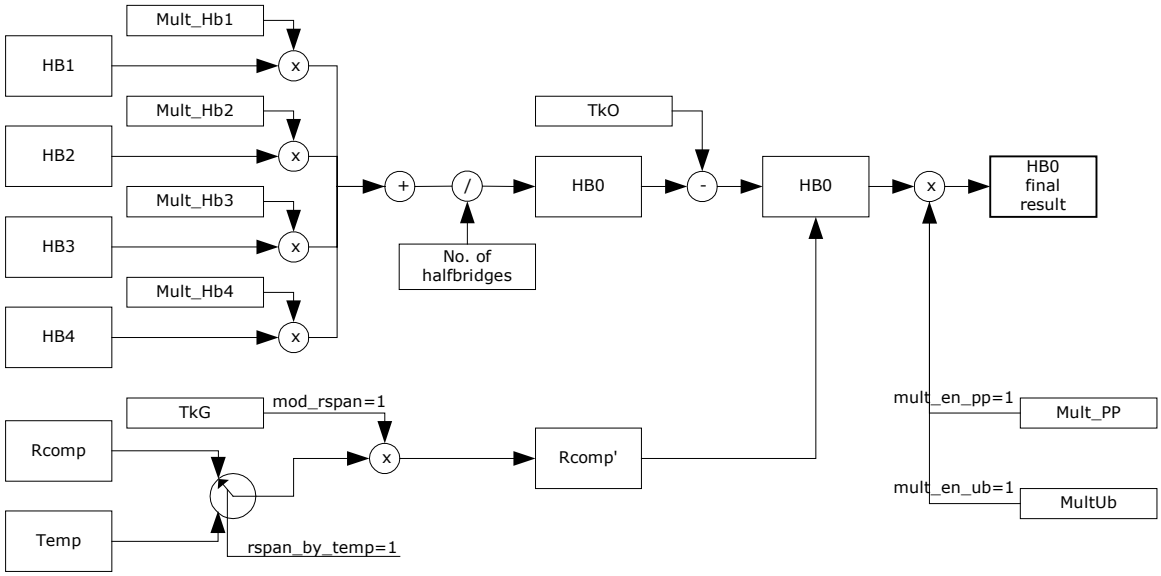
9.8 预处理

在测量的最后，转换器通过 ROM 内的代码对于测量还进行了一些预处理。它稳定的将校准过的以及拉伸过的测量结果存入 RAM 单元中。之后，如果 epr_usr_prg =1，那么 EEPROM 中的程序将被启动。

预处理的一些特殊性:

- 4 个半桥的结果是有 4 个分别的乘法因数的。这将提供了对于 4 驱衡器的分别半桥的软件校准。
- 如果传感器连接为半桥或者全桥模式。您需要选择相反的乘法因数。比如说假如您想给半桥结果乘以一个大于 1 的系数时，Mult_HB1 = 1 and Mult_HB2 = -1 或者 Mult_HB1 = 4 然后 Mult_HB2 = -4。
- 应变传感器和 span 补偿电阻是分开的。增益的补偿则可以通过软件进行。甚至温度测量可以代替 span 补偿电阻来对 span 进行补偿。仅通过软件方法可以将普通传感器提升到高质量的传感器。
- 被校准过的结果可以根据电池的电压进一步与校准系数相乘。这将提供电源抑制同时允许在没有电压调整情况下直接用电池供电。

图 39



一个简单的显示测量结果的程序如下:

```
ramadr    20      ;
move      x , r   ; 加载测量结果到 x-累加器
move      y , 2   ; 加载小数点的位置到 y-累加器
no2lcd            ; 7-短码的转换
newlcd            ; 刷新 LCD
clrwdt            ; 将看门狗定时器重置
stop              ; 停止 uC 工作
```

9.8.1 Gain 和 Offset 的温度补偿

如果在传感器桥臂上存在温度补偿电阻 (Rcomp 或 Rspan)，那么 PS08 可以测量这个阻值，然后通过数学算法来软件补偿。他们可以被应用于补偿整个电路的以及传感器本身的温度漂移。

有几种不同的方法来应用这些补偿电阻:

a) 如果补偿电阻已经与传感器很配。 那么无需额外调整温补电阻，就用它本身的值
b) 如果补偿电阻与桥臂比较匹配，但是桥臂有 offset，这个 offset 是可以通过软件去除掉的。那么对于 Rcomp 所引入的额外的温度漂移业可以被软件降低。
c) 如果能够温度箱里进行测试的话,那么就可以非常好的确定补偿系数 TK-Gain 和 TK-Offset。在这个情况下整个系统的补偿可以被大大的提高。比如我们的样秤，通过对于补偿系数的设置我们将 gain-drift 从 86.7 ppm/K 到 10 ppm/K 在 30K 的温度范围然后 offset-drift 从 290 nV/K 提高到 10 nV/K.

d) 如果传感器的增益误差已知 (比如在整个批次增益误差是一定的，但是是错误的值)那么可以直接通过 PS08 进行补偿无需再进行温度试验。

另外还需要注意的是, 通过 c)项还可以软件补偿一个并不是与桥臂很匹配的补偿电阻 Rcomp. 因此无需手动在硬件修正 Rcomp.

如果您想应用 PS08 温度校准的话,请联系我们，我们将会告诉您如果正确的进行温度补偿。

9.8.2 四驱秤的脚差纠正

有很多衡器比如人体秤有 4 个不同的传感器，每个传感器有一个半桥在上面。如果传感器的灵敏度有差异的话，如果重量不在中间放置的话，那么显示的重量在不同脚是有偏差的。那么 PS08 可以通过软件对于每个半桥分别进行调整，通过每个半桥的乘法因数(Mult_HB1 到 Mult_HB4). 简单的通过 4 个测量，就可以非常方便的对角差进行纠正。方法为一个标准重量依次序放在衡器的每个脚，然后再每个脚记录下来 HB0 到 HB4.
对于如何进行 Mult_HB1 到 Mult_HB4 测量结果的计算，请联系 acam 公司的工程师.

9.8.3 PS08 本身的 Gain-漂移

PS08 有非常低的 Gain 漂移，小于 1ppm/K. 那么 Gain 漂移的原因与 AD 转换器是不同的。因此我们在这个章节中给出一些背景知识，来进一步了解 PS08 的 gain 漂移，同时给出一些正确测量的提示。

背景: 在经典的 A/D 转换器应用中，放大器的电阻比需要非常精确的进行匹配。如果不匹配的话，会有比较大的 gain 漂移. 因为 PS08 与数模转换不同，应用的是时间数字转换。那么所得到的 gain 漂移是不同的，主要的 PS08 的 Gain 漂移是由于比较器电路所引起的。更具体一点说，就是由于比较器延迟在温度不同情况下不同所引起的。

我们降低比较器在温度范围内的这个漂移，我们推荐进行一些测量：

比较器电路中的电容应该为 2.2 – 2.7nF 的 COG 类型电容
(连接到 STOP, pin 40 的电容)

比较器电路中的电阻应该为 3.3 - 4.7kOhm
(连接到 STOP, pin 40 的电阻)

在寄存器 0 中选择 sel_compr[15:14] 到 4.1k 或者 7k
(在评估 labview 软件中这个设置可以在页 PS08 → Comparator → Comparator resistor value 中进行设置)

在寄存器 11 中设置 con_comp[1:0] 到'ON during Load'

[在 labview 评估软件中这个设置可以在页 PS08 → Comparator → Comparator control 中进行设置]

MULT_PP 的值

在上面所推荐的硬件中系统的整个增益误差在大约-4 ppm/K. 这是在生产过程中问鼎的并且与传感器的匹配无关的。这个所剩下的增益误差可以进一步被减小到< 1 ppm 通过选择正确的 MULT_PP 数值. 一旦在试验过程中将这个值确定下来后那么这个数值可以应用在整个系列的生产线当中。 一个好的 MULT_PP 值得范围是在 1.2 到 1.3 之间，与 cycle time 和充电电容有一些关系。

9.9 PS08 的高精度测量

PS08 的应用覆盖了很多应用并且在寄存器配置上给出了很多可能。因此比如说芯片可以很容易的配置为低功耗应用或者高频率测量应用或者高精度的测量应用。根据测量的应用不同，芯片需要被设置为适合的配置，同时电路也需要进行进一步调整。在这个子章节中我们将着重介绍如何将芯片进行正确的高精度测量的设置。

电路部分的调整:

线性稳压电源 在 3.3 到 3.6 V [不要开关电源]

对于 VCC_LOAD 的电解去藕电容为 >= 680µF

选择 C_{UCOMP1} 到 33µF 然后并联一个 2.2 uF 的陶瓷电容

增加 Cload 是放电时间在 110µs – 130µs 的范围, 比如在 350Ω传感器情况下应用 400nF COG 材料电容或者 CFCAP 系列电容器.

参数设置:

选择半桥的乘法因数> 1 [Mult_HB1..Mult_HB4]
[评估软件: 页 ALU → Multiplication Factor HB1 .. HB4]

在寄存器 3 中设置参数 ps_qziel[5:0] 到 33（十进制）.
[评估软件: 页 PS08 → PS08 Adjust 1]

在寄存器 1 中第 9 位设置参数 en_avcal 到 1
[评估软件: 页 ALU → 开启 16-次 TDC cal 值的平均]

在寄存器 10 中设置参数 mult_pp[7:0] 到一个经验证测出的数值（1.25）
[评估软件: 页 ALU → Multiplication Factor for Gain Correction]

在寄存器 2 中选择改变 avrate[23:14] 的值来调整到所需的刷新率
[评估软件: 页 PS08 → averaging rate]

一些提示:

将半桥以反方向连接，然后选择一个的乘法因数为正，另一个为负数。比如 Mult_HB1 = 4, Mult_HB2 = -4.
请注意, 平均率越高，那么精度也就越高。另外，输出刷新频率将会在加大平均率的情况下变低。
在高精度应用中我们推荐使用多层 PCB 或者至少两层板。应用多层 PCB 您可以将噪声降低，而且您可非常明显的看到精度的变化。
当然您也可以通过滚动平均/ SINC filter 的方法来增加精度。对于这一点我们提供了 SINC3, SINC4, SINC5

和 SINC8 在我们的评估软件中来看直接来进行滚动平均的效果。

9.10 PS08 与外部单片机连接

虽然 PS08 内部集成了一个强大的 24-bit 微处理器，但是这颗芯片同时也可以作为单一的转换器使用外接单片机。在这个情况下 PS08 工作在从模式下，单片机工作在主模式，并且通信协议为 SPI。

如章节 7 所描述的有很多 SPI 命令。通过读写以及一些 PS08 的特殊命令如 'Initreset' 或者 'Start_new_cycle' 芯片可以以 SPI 从模式工作.单片机初始化控制的基本步骤如下：

上电复位 → 配置寄存器 → 初始化 → Start_new_cycle

根据寄存器的设置有一点特殊的地方我们想指出：
配置寄存器的地址在这个情况下是不连续的。configreg_00 到 configreg_12 是连续的，从 RAM 地址的 48 到 60, 但是然后的 configreg_spec 是在 RAM 地址 64. 3 个中间丢失的寄存器 configreg_13 到 configreg_15 并不需要被配置，因为他们的内容是显示 LCD 显示值，这个值是时时在变化的。
默认的 configreg_spec 设置可以在 3.4.1 配置寄存器当中找到。

10 振荡器

PS08 有一个内部的低电流 10kHz 振荡器用来作为时钟基础以及在 streched 模式和测量范围 1 的情况下定义 cycletime 的。
另外, PS08 提供一个为外接 4Mhz 陶瓷晶振的驱动。这个晶振是用来做时间测量以及定义测量范围 2 的 Cycletime。它大概需要 130 µA @ 3.0 V。
配置: 寄存器 3, Bits 17 - 19: sel_start_osz
0 = 关闭晶振
1 = 持续开启晶振
2 = 在开启后100 µs 延迟后开始测量
3 = □□□□200 µs □□□□□□□□
4 = □□□□300 µs □□□□□□□□
5 = □□□□400 µs □□□□□□□□
6 & 7 不需要联系
□□□ 2, Bit 0: auto10k

这个晶振可以持续开启或者只在测量当中开启，包括达到满振幅的启振时间(sel_start_osz[2:0])。4MHz 晶振的启振时间大约为 50µs 到 100µs 与电源电压有一定关系。

自动校准:

内部的 10 kHz 晶振可以自动通过 4MHz 晶振进行校准。因为频率是受到温度和电压影响的，当应用 10 kHz 晶振作为输出刷新率以及采样的频率的时间转换计数时，以及当在 Stretched 模式下作为 Cycletime 的时候，是会受到很大影响的。因此我们推荐应用自动校准功能，也就是设置 auto10k = 1.

注:
不推荐在 stretched 单一转换模式中应用自动校准!

11 电压测量

一个内部带隙电压参考用于测量电压的大小。 这个测量频率为 40 次每秒钟。结果被存放在地址 25 的 RAM 里面，UBATT。 他是通过公式 电压 = 2.0 V + 1.6 V* UBATT/64 计算的。

结果可以被用作

- 低电池电压检测: 电压水平在控制寄存器 low_batt[2:0]被设置

low_batt	0	1	2	3	4	5	6	7
电压水平 [V]	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9

在状态寄存器中的标志 flg_ub_low 说明是否电压低于所设置的水平。

-电源抑制: 测量电压可以用于纠正电压的增益依赖特性。这是通过设置控制寄存器位 mult_en_ub = 1 和 mult_ub[7:0] 来开启的。应变测量的结果将会根据 HB = HB/(1 + UB*[-128 ... 127]/2^21)被校正。

- EEPROM 保护: 当电压低于 2.4 V 时 自动 EEPROM 写操作 (putepr) 是禁止的。 这将保护 EEPROM 不会产生数据崩溃。

注意: 如果供电电压在 2.1V 以下，测量将回不正确 (显示数值太高) 那么测量的结果不能被采用. 在 1.5 V 系统中是不能够测量 PS08 的电源的。

12 Auto-on

PS08 可以在自动开启模式中运行。在休眠模式的情况下 PS08 以 avrate = 1 (最小精度) 测量，测量刷新率也是非常低 (比如 1 Hz).在这种配置情况下系统电流消耗可以降低到 2µA.如果在衡器的称重面上有重量被检测到，那么平均率 avrate 和测量频率通过软件内部再次配置 avrate 和 tdc_conv_cnt 而增加到预期值。非常大的优点就是这样 LCD 可以在没有任何延迟的情况下很快显示一个正确的结果。

13 测量范围 1

在这个模式下 TDC 的测量范围被减小到 15 µs. 因此放电时间必须要减小到< 10 µs.

配置: 寄存器 1, Bit 18: messb2 = 0

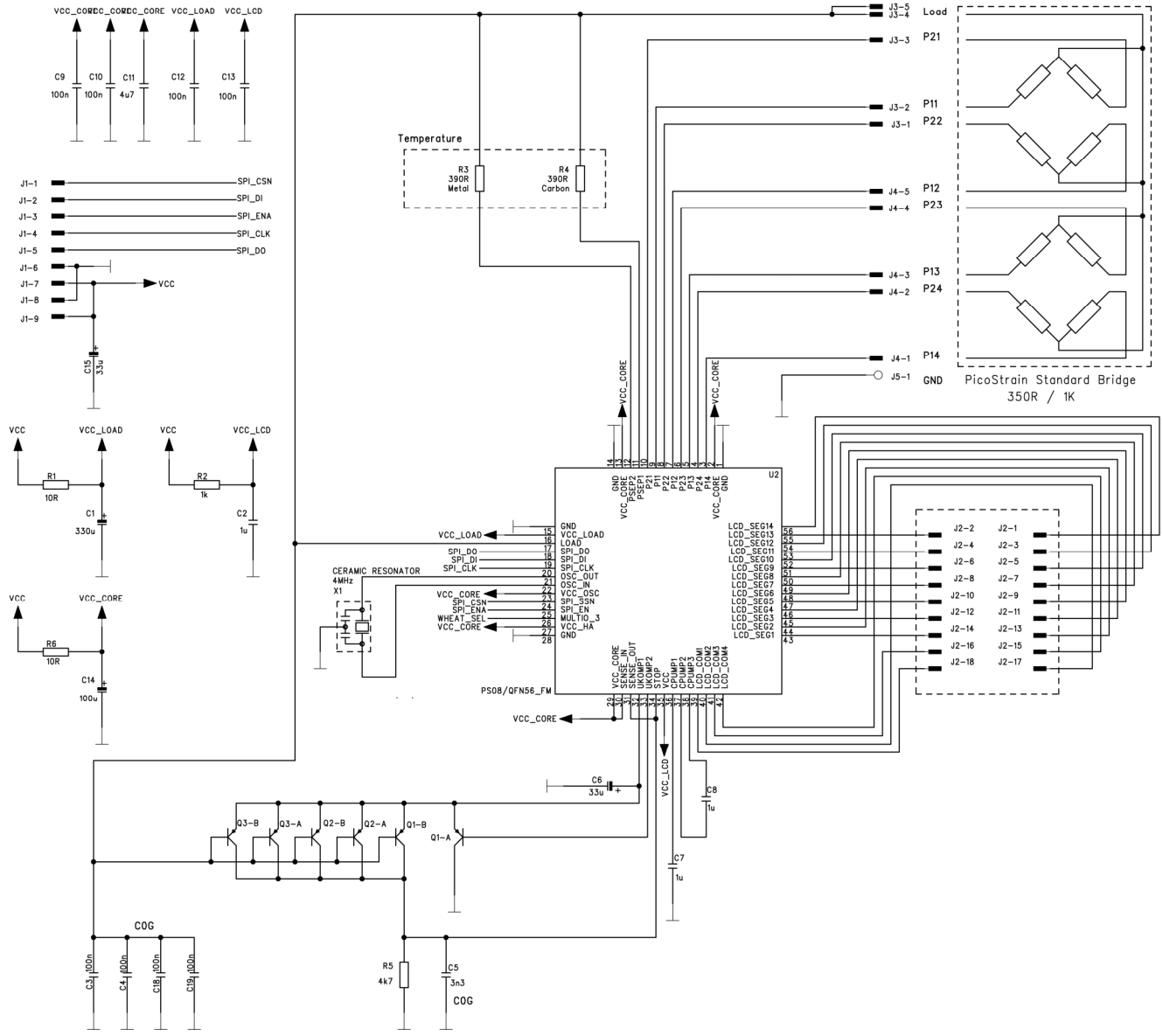
在测量范围 1 中 cycle time 是通过 10 kHz 晶振时钟产生的。

优点: 无需外部的 4 MHz 晶振
缺点: 精度下降。这个模式可以应用于 2000 分度(10000 内部分度).
电流消耗比不带测量范围 1 的情况要高，因为 TDC 高速时钟是在整个放电时间测量过程中开启的。

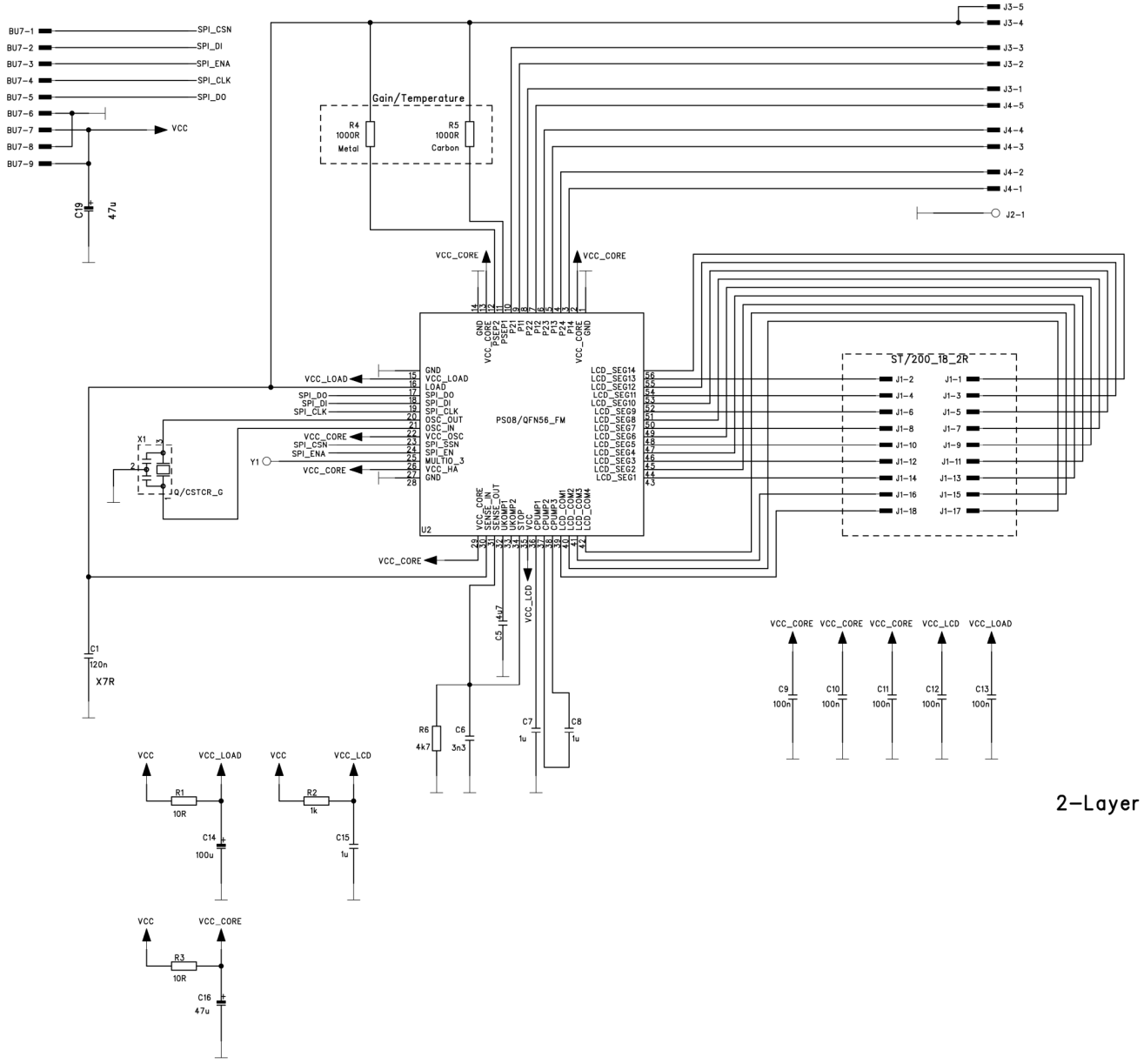
10 kHz 晶振不能够被自动校准 (auto10k =! 0).

14 样例电路

14.1. 高精度应用电路



14.2 低精度应用电路


Gain/Temperature
R4 1000R Metal
R5 1000R Carbon
PS08/QFN56_FW
U2
ST/200_18_2R
JQ/CSTCR_G
X7R
2-Layer

15 已知的 Bugs

1. I/O Pins	In case mod_rspan is set to 1 there might be situations where the status of the input pins regarding rising or falling edge is not correctly updated. Workarround: There is a software workarround availabe from acam as an additional header file. Please contact acam.

最后更改

16.04.2008 第一次编辑
14.05.2008 v0.2: p.22: cytime
21.05.2008 v0.3: p.49 new opcode New_LCD
11.07.2008 v0.4: Wiring diagrams with added Gnd lines

联系方式

Headquarter Germany	acam-messelectronic gmbh	Am Hasenbiel 27 76297 Stutensee-Blankenloch	Tel: +49 (0) 7244 7419-0 Fax: +49 (0) 7244 7419-29 support@acam.de www.acam.de

European Distributors

Belgium (Vlaanderen)	CenS (Micro) Electronics BV.	PO Box 2331/ NL 7332 EA Apeldoorn Lamfe Amerikaweg 67 NL 7332 BP Apeldoorn	Tel: +31 (0) 55 3558611 Fax: +31 (0) 55 3560211 info@censelect.nl www.censelect.nl
France	microel (CATS S.A.)	Immeuble "Oslo" - Les Fjords 19, avenue de Norvège Z.A. de Courtaboeuf - BP 3 91941 LES ULIS Cedex	Tel. : +33 1 69 07 08 24 Fax : +33 1 69 07 17 23 commercial@microel.fr www.microel.fr
Great Britain	2001 Electronic Components Ltd.	Stevenage Business Park, Pin Green Stevenage, Herts SG1 4S2	Tel. +44 1438 74 2001 Fax +44 1438 74 2001 a.parker@2k1.co.uk www.2k1.co.uk
Hungary	ChipCAD ELEKTRONIKAI DISZTRIBUCIÓ KFT	Tuzolto u. 31. 1094 BUDAPEST	Tel: +36 231 7000 Fax: +36 231 7011 Email: szfarkas@chipcad.hu www.chipcad.hu
Italy	DELTA Elettronice s.r.l	Via Valpraiso 7/A 20144 Milano	Tel: +39 02 485 611 1 Fax: +39 02 485 611 242 email: afrigerio@deltacomp.it www.deltacomp.it
Netherlands	CenS (Micro) Electronics BV.	PO Box 2331/ NL 7332 EA Apeldoorn Lamfe Amerikaweg 67 NL 7332 BP Apeldoorn	Tel: +31 (0) 55 3558611 Fax: +31 (0) 55 3560211 info@censelect.nl www.censelect.nl
Poland	W.G. Electronics Sp.z o.o.	ul. Modzelewskiego 35 02-679 WARSZAWA	Tel: +48 22 847 9720, 847 9721 Fax: +48 22 647 0642 Email: tgornicki@wg.com.pl www.wg.com.pl
Switzerland	Computer Controls AG	Neunbrunnenstr. 55 8050 Zürich	Tel.: +41-1-308 6666 Fax: +41-1-308 6655 email: roeschger@ccontrols.ch www.ccontrols.ch
Russia	Galant Electronics, Ltd.	100, Prospekt Mira, Moscow, 129626, Russia	Tel\Fax: +7-495-987-42-10, Tel: +7-095-107-19-62 Mobile +7-916-993-67-57 Email: leonid-k@galant-e.ru www.galant-e.ru

American Distributors

United States of America	Transducers Direct, LCC	264 Center Street Miamiville, Ohio 45147	Tel: 513-583-9491 Fax: 513-583-9476 Email: sales@acam-usa.com www.acam-usa.com

Asian Distributors

India	Brilliant Electro-Sys. Pvt. Ltd.	4, Chiplunker Building, 4 Tara Temple Lane, Lamington Road, Bombay – 400 007	Tel: +91 22 2387 5565 Fax: +91 22 2388 7063 www.brilliantelectronics.com besimpex@vsnl.net
Israel	ArazimLtd.	4 Hamelacha St. Lod P.O.Box 4011 Lod 71110	Tel: 972-8-9230555 Fax: 972-8-9230044 email: info@arazim.com www.arazim.co.il
Japan	DMD–Daiei Musen Denki Co., Ltd.	10-10, Sotokanda, 3-Chome, Chiyoda-Ku Tokyo 101-0021	Tel: +81 (0)3 3255 0931 Fax: +81 (0)3 3255 9869 sales@daiei-dmd.co.jp www.daiei-dmd.co.jp
P.R. China	Broadtechs Technology Co. Ltd.	3C JinHuan Building, 489 Xiang Yang Road South Shanghai, 200031	Tel.: +86-21-54654391 Fax: +86-21-64454370 Email: info@acam-china.com www.acam-china.com
	Shenzhen SECOM TELECOM Co., Ltd.	Headquarter: 32/F, Block A, ShenFang Plaza, No. 3005 Renmin Nan Rd. Shenzhen 518001 Nanjing Office: Beijing Office: Qingdao Office: Shanghai Office: Chengdu Office: Wuhan Office: Xi'An Office: Xiamen Office:	Tel.: +86 755 25155888 Fax: +86 755 25155880 Email: zorro_huang@secomtel.com www.secomtel.com Tel.: +86 25 84552900 Tel.: +86 10 82336866 Tel.: +86 86 532 85899132 Tel.: +86 21 52371820 Tel.: +86 28 82981751 Tel.: +86 27 87322726 Tel.: +86 29 88323435 Tel.: +86 592 5806950
South Korea	SamHwa Technology Co., Ltd.	#4 4F Kyungwon building, 416-6 Jakjeon-dong GYEYANG-GU, INCHEON 407-060	Tel: +82 32 556 5410 Fax: +82 32 556 5411 www.isamhwa.com minjoonho@isamhwa.com